Enhancing feedback process scheduling via a predictive control approach

Alessandro Vittorio Papadopoulos, Martina Maggio, Sara Negro, Alberto Leva

Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio, 34/5 - 20133 Milano, Italy

Abstract: In some recent papers it was shown that preemptive process schedulers in multitasking operating systems can be viewed, and above all *designed*, as discrete-time feedback controls with very simple (I- or PI-type) regulators, yielding significant advantages over classical scheduling policies as for time complexity and parameter interpretability. In this work, the same problem is tackled with a predictive control approach. Doing so allows to release some simplification hypotheses of the mentioned papers and to improve the achieved solutions' flexibility, widening the application possibilities at an affordable additional cost.

Keywords: feedback scheduling; control of computing systems; predictive control.

1. INTRODUCTION

Many problems related to computing systems are nowadays being recognised and tackled as *control* ones, see, e.g., Hellerstein et al. (2004) and some papers quoted therein. Along such a reasoning, in recent years many contributions were proposed in different areas, from reliability (Kreidl and Frazier, 2004) to security (Dantu et al., 2007), software testing (Bayan and Cangussu, 2008), thread-pool management (Hellerstein et al., 2009), and more.

In such a *scenario*, a notable interest is encountered by the problem of scheduling in multitasking environments, as testified by Abeni et al. (2002); Palopoli et al. (2003), where a *process scheduler* allocates the CPU usage to the pool of running processes, to guarantee properties like fairness, responsiveness, and so forth. Feedback-based techniques have been applied to the scheduling problem to deal with uncertainties and disturbances, such as the behaviour of the processes and the availability of resources (shared memory, peripherals, and so on). This has led to the so-called "feedback scheduling" research topic, in the sense clarified by Xia and Sun (2006) for operating systems.

However, in virtually the totality of the feedback scheduling literature, the idea is (concisely) to "close some loop around an *existing* scheduler", see e.g. Lu et al. (1999, 2002). Since said scheduler was typically conceived as an algorithm, it is most often (not to say, always) unnatural to model it as a dynamic system. This hinders the use of simple - thus powerful - control synthesis and analysis formalisms.

The research to which this article belongs, takes completely different an attitude. Instead of acting on the scheduler that is already present in the considered system, the idea here is to completely *replace* that scheduler. Correspondingly, instead of writing a model to reflect the *existing* scheduling algorithm, the *modus operandi* is to have a *new* algorithm emerge from the digital realisation of a controller model (Maggio and Leva, 2010).

The above idea was recently put at work in Leva and Maggio (2010), where feedback scheduling in a uniprocessor system was realised by a cascade structure with discrete-time I- and PI-type controllers. The solutions shown in that paper outperform classical policies such as the Round Robin, the Earliest Deadline First, and so forth: there is a very moderate (if any) additional computational burden, but the obtained results (in terms e.g. of fairness or deadline misses) are significantly better, as proven in practice by a microcontroller-based implementation (Maggio et al., 2010).

In this work, a predictive controller is used to address the same problem of the quoted papers, allowing to release some of the introduced simplifications, and also to sketch out some interesting developments for future research.

2. BRIEF REVIEW OF PREVIOUS RESULTS

The starting point for this work is Leva and Maggio (2010), where - as herein - a single-processor multitasking system with a preemptive scheduler is considered, and the N processes to be scheduled are activates in a Round Robin fashion, assigning them a certain amount (possibly zero) of CPU time. Feedback is introduced in that said CPU time amounts come from the computation performed by a controller. To briefly review the matter, define the "scheduling round" as the time between two subsequent scheduler interventions, and let $\tau_p(k) \in \mathfrak{R}^N$, $\tau_r(k) \in \mathfrak{R}$, $b(k) \in \mathfrak{R}^N$, $\delta b(k) \in \mathfrak{R}^N$ represent, respectively,

- the CPU times *actually* allocated to the processes in the k-th scheduling round,
- the *actual* duration of the k-th round,
- the CPU times or *bursts* assigned to the processes at the k-th round,
- the disturbances possibly acting on the scheduling action during the k-th round, that appear as a difference between the burst and the CPU time actually consumed by the process.

Denoting by t the total time *actually* elapsed from the system initialisation, the scheduling problem can be addressed by

describing the controlled system (the process pool) with the simple model

$$\begin{cases} \tau_p(k) = b(k-1) \\ \tau_r(k) = r_1\tau_p(k-1) \\ t(k) = t(k-1)+r_1\tau_p(k-1) \end{cases} \quad (1)$$

where r_1 is an all-ones row vector of length N. The scheduler is then synthesised as a cascade controller, where the internal loop manages the CPU time distribution among the processes within the round, and the external one controls the time between two subsequent scheduler interventions (i.e., the desired "round duration").

Indicating with τ_r° said required round scheduling duration, and with

$$\theta_p^\circ \in \Re^N, \quad \theta_{p,i}^\circ \geq 0, \quad \sum_{i=1}^{N} \theta_{p,i}^\circ = 1 \quad (2)$$

the vector containing the required CPU time fractions to be allocated to each process, it is quite intuitive to see that virtually any specification on fairness, tardiness, and so forth, can be expressed in terms of the two references τ_r° and θ_p° above. The "scheduler as controller" scheme is thus represented by Figure 1, where an appropriate choice of the R_r and R_p regulators allows to attain the round duration set point, and the desired CPU percentages.

In Leva and Maggio (2010), the important remark is made that model (1) is diagonal. Hence, employing as R_p in the internal loop CL_1 of Figure 1 a diagonal integral controller *with all the gains equal*, i.e., computing the i-th burst $b_i(k)$ as

$$b_i(k) = b_i(k-1) + k_I\left(\tau_p^\circ(k-1) - \tau_p(k)\right), \quad (3)$$

where k_I is the mentioned value of all integral gains, presents to the regulator R_r of the external loop CL_2, controlling system S_2 by introducing the additive "burst correction" b_c, the very simple (discrete-time) transfer function

$$\frac{T_r(z)}{B_c(z)} = \frac{k_{pi}}{z(z-1)} \quad (4)$$

where k_{pi} is the gain of all the internal controllers and, notice, is independent of θ_p°. The design of R_r too is thereby straightforward, and a PI controller proves sufficient. The results achievable with the "I+PI" scheme (i.e., when R_p is a diagonal I controller and R_r a SISO PI) are exemplified in Figure 2, that reports the round duration τ_r versus its set point τ_r° and the additive correction b_c exerted by R_r (top left), the process CPU use τ_p versus its set point τ_p° (top right), the same data as in the top right plot but vertically split for the individual processes to evidence arrivals and terminations (bottom left), and finally the process bursts normalised between two conveniently chosen saturation values (bottom right). Note, incidentally, that model (1) is valid whatever burst (feedback) calculation scheme is adopted.

The simulations presented here were obtained by replicating the scheduling algorithms in the Scilab environment (Campbell et al., 2006). Such a choice is motivated by the innovative character of the new proposed algorithms, that can be hardly represented in the typical environments used for schedulers' assessment.

In said simulations, the applied *stimuli* are (a) some modifications of the required CPU distribution, to show that the basic goal of the scheduler is attained, (b) some modifications to the pool of processes, to prove that re-initialising the controller - a matter omitted here for brevity - is possible and quite straightforward, (c) some process blockings, to test the treatment of that case by the various solutions, and (d) some modifications of the desired round duration, to illustrate that the system responsiveness can actually be changed on-line thanks to the nested loop structure of Figure 1.

Referring the reader to the quoted works for the details here omitted, two basic questions remain open. First, is it possible to improve results by adopting (slightly) more complex regulator blocks? Second, and more relevant, is it possible with analogous complications to release the assumptions that the inner controllers have equal gains, in the absence of which the design of the external one may sometimes be critical? The rest of this manuscript aims at answering those questions, by means of predictive control.

3. PREDICTIVE CONTROL FOR FEEDBACK SCHEDULING – A MOTIVATION

As noticed in Leva and Maggio (2010), with the PI-based structure adopted therein, two main problems stand open. The first one is the significant complication introduced by possibly different gains in the inner I loops. The second one is that if some process gets blocked (i.e., continues refusing to execute and returning the CPU to the scheduler immediately at the beginning of its burst) this pushes the controller into a saturation state, that is merely managed by standard antiwindup.

Both problems appear to be manageable by adopting for the external loop controlling S_2 in Figure 1 a predictive structure, and theoretically such a solution should yield some improvement. In fact, in the considered control system, cross-coupling is introduced by the outer loop only, and both the mentioned problems can be seen as an alteration of the dynamics seen by the outer controller. Applying a predictive controller with suitably extended prediction and control horizons should in principle allow to cope with such modification better than a mere integrator.

Also, when managing blocked processes, the PI-based structure inherently preserves the round duration. In such a case one may want the CPU time assigned to each process to remain the same (with some idle time corresponding to the blocked processes) or to be re-distributed among non blocked processes (increasing the CPU time percentage of each non blocked process). A predictive controller allows to manage such choices in more neat a way than control structures like those presented in Leva and Maggio (2010).

More in general, although the matter is not treated here, problems analogous to the scheduling one are quite naturally formulated as optimisation ones, where (strictly) there is no set point to follow, but the system has to optimise some performance metrics while at the same time strictly remaining within certain constraints or bounds—bandwidth allocation is a natural example to think about. Predictive control techniques are apparently more suited for such kind of desires than PI-based control, whence the interest for their use in the addressed problem in view of devising more general a formalism.

4. THE PROPOSED CONTROL SCHEMES

4.1 Foreword

In this work, controller R_r in Figure 1 is replaced by a RHPC (Receding Horizon Predictive Controller), while for R_p the

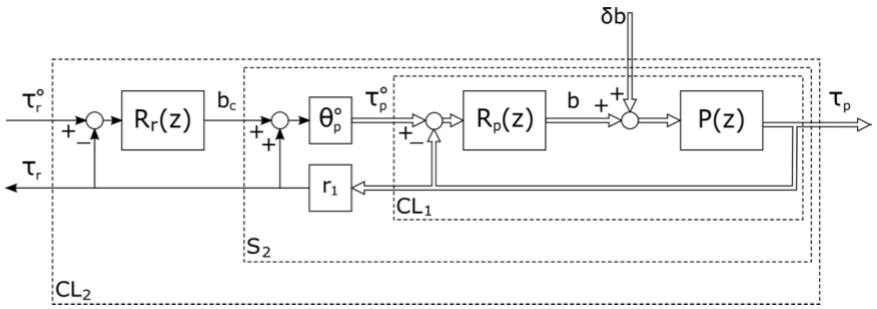

Fig. 1. The proposed "scheduler as controller" scheme as per Leva and Maggio (2010); double-line arrows denote vector signals.

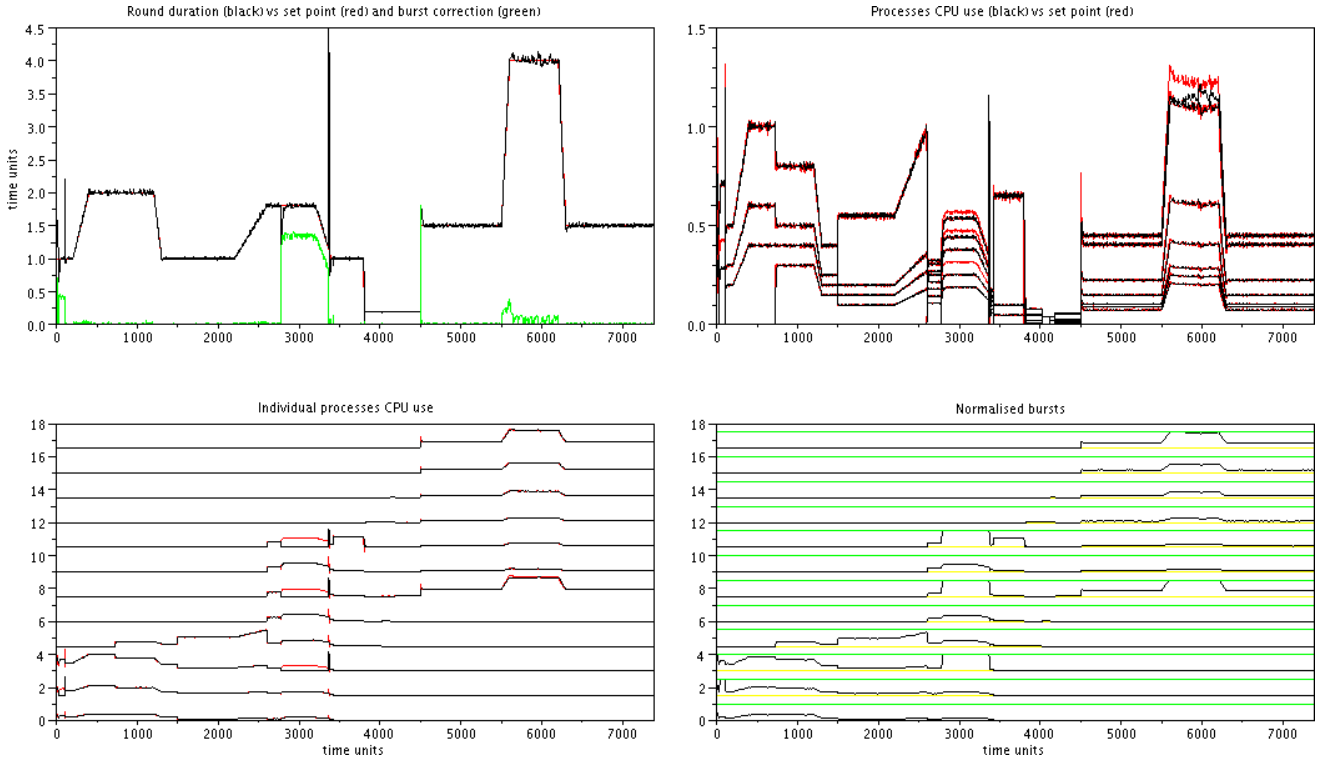

Fig. 2. Example of the results achievable with the I+PI structure.

solution of Leva and Maggio (2010) is maintained. Referring the reader e.g. to Camacho and Bordons (2004) for further information on RHPC, this section just illustrates the specific choices adopted for its application in the addressed context.

First, reconsider the scheme of Figure 1 with the inner loops closed. If all the I gains are equal and no process is blocked, the transfer function from $B(z)$ to $T_r(z)$ is given by (4). Suppose now that said gains are not equal anymore: expressing the i-th gain as $\beta_i k_{pi}$, and representing a blocked process by zeroing its gain - i.e., replacing the first equation of (1) with $\tau_p(k) = \Gamma b(k-1)$ where $\Gamma := diag\{\gamma_i\}$ is a matrix with one or zero diagonal elements (zero means a blocked process) - the same transfer function becomes

$$\frac{T_r(z)}{B_c(z)} = \frac{k_{pi}}{z(z-1)} \frac{\Delta_N(z)}{\Delta_D(z)} \tag{5}$$

where $\Delta_N(z)$ and $\Delta_D(z)$ are polynomials in z of degree N, the expression of which is lengthy and thus omitted. Suffice to say that the multiplicative term $\Delta_N(z)/\Delta_D(z)$ can alter the dynamics of (5) significantly with respect to (4).

Suppose then to select as R_p a diagonal integral regulator with *different* gains $\beta_i k_{pi}$, see above, which corresponds in the state-space to

$$\begin{aligned} A_{R_p} &= C_{R_p} = I_{N\times N} \\ B_{R_p} &= k_{pi}[\beta_1 \ \beta_2 \ \dots \ \beta_N] I_{N\times N} \\ D_{R_p} &= 0_{N\times N} \end{aligned} \tag{6}$$

where $I_{N\times N}$ and $0_{N\times N}$ are respectively the identity and the zero matrix of dimensions $N \times N$. Now, denote by τ_r° the required scheduling round duration, and let the required CPU time fractions to be allotted to each process be defined as per (2). If τ_p° were chosen as $\theta_p^\circ \tau_r^\circ$, then CL_1 would control both the CPU distribution and the round duration, but there would be two problems. First, the dynamics of those two controls would be ruled by the same eigenvalues, which can be inadequate in some cases. For example, one may want the CPU distribution to move smoothly from one situation to another, but the round duration to respond very quickly to its set point. Second, and more serious, as anticipated, if some processes are blocked, the round duration set point cannot be attained. Introducing the information of blocked processes, the system denoted in Figure

1 by S_2 assumes a switching nature, having the dynamic matrix

$$A_{S_2} = \begin{bmatrix} 0_{N\times N} & B_\Gamma C_{R_p} \\ B_{R_p}(\theta_p^\circ r_1 - I_{N\times N}) & A_{R_p} \end{bmatrix} \quad (7)$$

where

$$B_\Gamma = \gamma I_{N\times N} = \begin{bmatrix} \gamma_1 & 0 & \dots & 0 \\ 0 & \gamma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \gamma_N \end{bmatrix} \quad (8)$$

is the (switching) dynamic matrix for the processes to be controlled containing the information on the blocked ones, i.e., $\gamma_j = 1$ if the process is active and $\gamma_j = 0$ if the process is blocked.

Based on the state-space model just sketched, a "vanilla" RHPC control law can be designed disregarding its switching nature, i.e., acting as if no blocking ever occurred. The resulting control law is

$$b_c(k) = b_c(k-1) + \Delta b_c(k) \quad (9)$$

where the $\Delta b_c(k)$ is computed as

$$\Delta b_c = \overbrace{[1\ 0\ \dots\ 0]}^{N_c}(\Phi'\Phi + \bar{R})^{-1}\Phi'(\bar{T}_r^\circ \tau_r^\circ(k) - Fx(k)) \quad (10)$$

and Φ and F depend on the model matrices as explained e.g. in Camacho and Bordons (2004), $\bar{R}$ is a diagonal matrix in the form that $\bar{R} = wI_{N_c\times N_c}(w \geq 0)$ where w is used as a tuning parameter for the desired closed-loop performance, and finally $\bar{T}_r^\circ$ is an all-ones column vector of length N_p. Of course, in the case of "symmetric desires", where the integral gains are equal, and if no information about the blocked processes is available (or equivalently, the processes are always considered not to be blocked), then the resulting SISO system with input b_c and output τ_r has the transfer function (4), which yields the state-space model

$$A_{S_2} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \quad B_{S_2} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C_{S_2} = [k_{pi}\ 0], \quad D_{S_2} = 0 \quad (11)$$

It was verified (details are inessential here) that the so obtained control is equivalent to the I+P solution prosed in Leva and Maggio (2010). More interesting are the *additional* possibilities of the RHPC, explored below.

4.2 The "asymmetric desires" case

Processes running on a generic system have very different purposes, requirements, and behaviours, yet they have to coexist all together on the same computing system. This leads to the fact that the OS scheduling algorithm must fulfil several conflicting objectives: fast process response time, good throughput for background jobs, avoidance of process starvation, reconciliation of the needs of low- and high-priority processes, and so on. These objectives cannot be easily satisfied with classical approaches.

In the mainstream computer science literature, when speaking about scheduling, processes are traditionally classified as "I/O-bound" or "CPU-bound". The former type makes heavy use of I/O devices and spends much time waiting for I/O operations to complete; the latter type are number-crunching applications that require a lot of CPU time.

Alternatively, another classification distinguishes three classes of processes: the interactive ones (processes which interact constantly with their users, and therefore spend a lot of time waiting for key-presses and mouse operations), batch ones (processes which do not need user interaction, and hence often run in the background) and real-time processes (processes which should never be blocked by lower-priority processes, have a short response time and, most important, exhibit for such response as low a variance as possible).

Whatever is the case, different kind of processes which must run on the same computing system are treated as different entities. This explains in practice why nowadays there exist a variety of *ad hoc* scheduling techniques for specific classes of application, e.g., real-time application scheduling. This is far from being a general approach, apparently.

The main advantage in a control theoretical standpoint is that, such a classification does not lead to different scheduling algorithms, but it is easier to generalise the case in which the processes have different features and must be treated in different ways, according to their specifications and nature.

In the previous section, a symmetric internal loop CL_1 was considered (see Figure 1), making the assumption that all the processes can be treated in the same manner. A more general approach is to use different weights for the different processes' I controllers in the internal loop, in order to distinguish the low-priority processes from the high-priority ones by means of lower or higher gains respectively, leading to different response times.

The I+PI approach can still lead to good results, from the above point of view, as exemplified in Figure 3, but when the internal loop has different regulators its design is not so simple as it is in the symmetric case, and in particular, it is not independent of θ_p° anymore, and if not carried out properly, can also produce unstable behaviours. Since only the external loop is considered, Figure 3 and the subsequent analogous ones are organised in the same way as the top left plot of Figure 2.

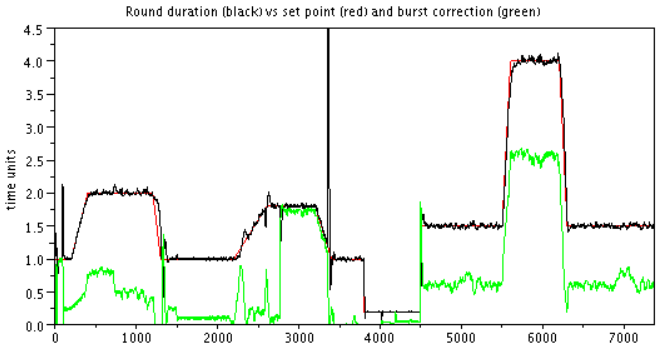

Fig. 3. The round duration controlled with an external PI controller and with different gains in the internal loop.

With the RHPC it is on the contrary quite straightforward to design the controller by resorting to (7) and merely releasing the symmetry assumption. Analysing the corresponding simulation plots of Figure 4, one can see how the predictive approach leads to substantially analogous results with respect to the I+PI, although its design is simpler.

4.3 Accounting for blockings: an actuation problem

Blocked processes may lead to undesired behaviour of the system. Both with the PI (see Figure 2) and with the RHPC, we have reached the goal of rejecting such disturbances and

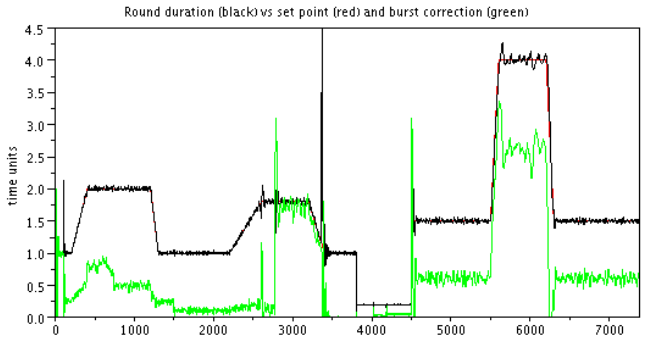

Fig. 4. The round duration controlled with an external RHPC controller and with different gains in the internal loop.

of following the set point signal. However, there is an open *actuation* problem: what happens to the burst assigned to each process when one or more processes are blocked? How should the actuator distribute the CPU time among the processes within the round?

With the I+PI approach, there are two main solutions of this actuation problem. The first solution is to maintain the round duration constant, as if all processes were scheduled, and the time that should have been allotted to the blocked processes were considered "idle time". This idea implies an action on the actuators, not on the controller, in that if a process returns the CPU before the expiration of its burst, the system (more precisely, its actuating part) just has to wait till the allocated burst is exhausted. The second solution is to re-distribute the time of the round duration among the active non blocked processes. As in the first case, the round duration remains constant, but the percentage of CPU time of each process changes.

The use of the RHPC implies new possibilities in this *panorama*. For instance, one can open the external loop, releasing the round duration control, which may change at each scheduling round. The round duration becomes the sum of the bursts assigned to the processes at the k-th round, except that assigned to the blocked processes. The information about blocked processes at the last step can be used by RHPC to predict the output. For instance, if we have 10 processes to be scheduled, a round duration of 10, each process has a burst of 1 unit and one process is blocked, the round duration becomes 9.

This solution is much better than those offered by the simple PI control structure. For example, in the case of a CPU for a desktop PC, it may not be so dramatic if the CPU is idle because of a blocked process, but in the case of a web server or a router, where the waste of CPU time is a greater damage, it would be better to use all the available CPU time or bandwidth, or, in general, all the available resources.

Also, with RHPC, it is resonably simple to account for process blockings by simply having them detected by the operating system (details on that would stray from this work but pose non serious technical problems), and change the predictive model used in the controller by acting on (8). The so obtained results are shown in Figure 5, organised in the same way as Figure 2.

5. DISCUSSION AND PERSPECTIVES

In the context of this work, there is a huge interest in designing applications according to specific requirements, and sometimes in strictly limiting the behaviour under certain constraints or bounds. Predictive control techniques offer a means to satisfy such desires, allowing more flexibility in the transitory management.

Furthermore, in the scheduling context, with the PI regulation structure, the blocked processes management corresponds to the switching of the system and does not impact on the round duration which remains the same, while the CPU time assigned to each process may remain the same (with some idle time corresponding to the blocked processes) or may be re-distributed among non blocked processes (increasing each CPU time percentage). A predictive control approach can lead to better solutions for the blocking processes problem.

One could object that the results obtained with the I+P(I) scheme are quite good, so it is not convenient to use a more complex control technique. However, the simple I+PI structure is not *scalable*, in the sense that if we decide to use different weights on each computed error, the model becomes much more complicated and it is no longer reducible to a second order model, i.e., denoting by N the number of the running processes, in the worst case (a different weight for each process) the model becomes of order $2N$. Thus, in this case, it is convenient to use a more complex (predictive) control technique, which allows to specify different weights, with the same computational complexity.

In other words, to summarise the reported (brief) comparison of I+PI and RHPC, one can state that the former is tendentiously simpler to set up but less scalable. As such, the I+PI scheme is a good solution for "symmetric" problems, i.e., where the error weights can be taken equal. On the other hand, in situations characterised by asymmetry on the desires, it is more convenient to use predictive schemes such as the RHPC one just envisaged and exemplified. In fact such schemes, although they have greater computational complexity, are guaranteed to be successful in almost each situation.

6. CONCLUSIONS AND FUTURE WORK

This work is part of a long-term research aimed at the design of computing system (and more specifically, operating systems) critical parts, in the form of feedback controllers.

Previous papers have shown that for process scheduling, very simple control schemes can serve the duty under quite loose simplifications. In this manuscript, some slightly more complex solutions were proposed, based on a predictive approach, and leading to some advantages in terms of generality, at a moderate additional cost.

As such, it is even more firmly proven that the control theory provides a wealth of tools and methods, the majority of which to date unexploited, for the design of computing system—provided that the "correct" method is chosen for each problem, i.e., in turn, that the classification of said problems is faced with a system-theoretical attitude. Future work will further investigate so vast and promising a field.

In addition, predictive-like techniques are being applied to more complex problems, where substantial model uncertainty and/or variability exists, contrary to the case addressed herein. Encouraging results are already emerging from the use of predictive control in conjunction with on-line parameter identification and adaptation, thanks also to the fact that computer systems prac-

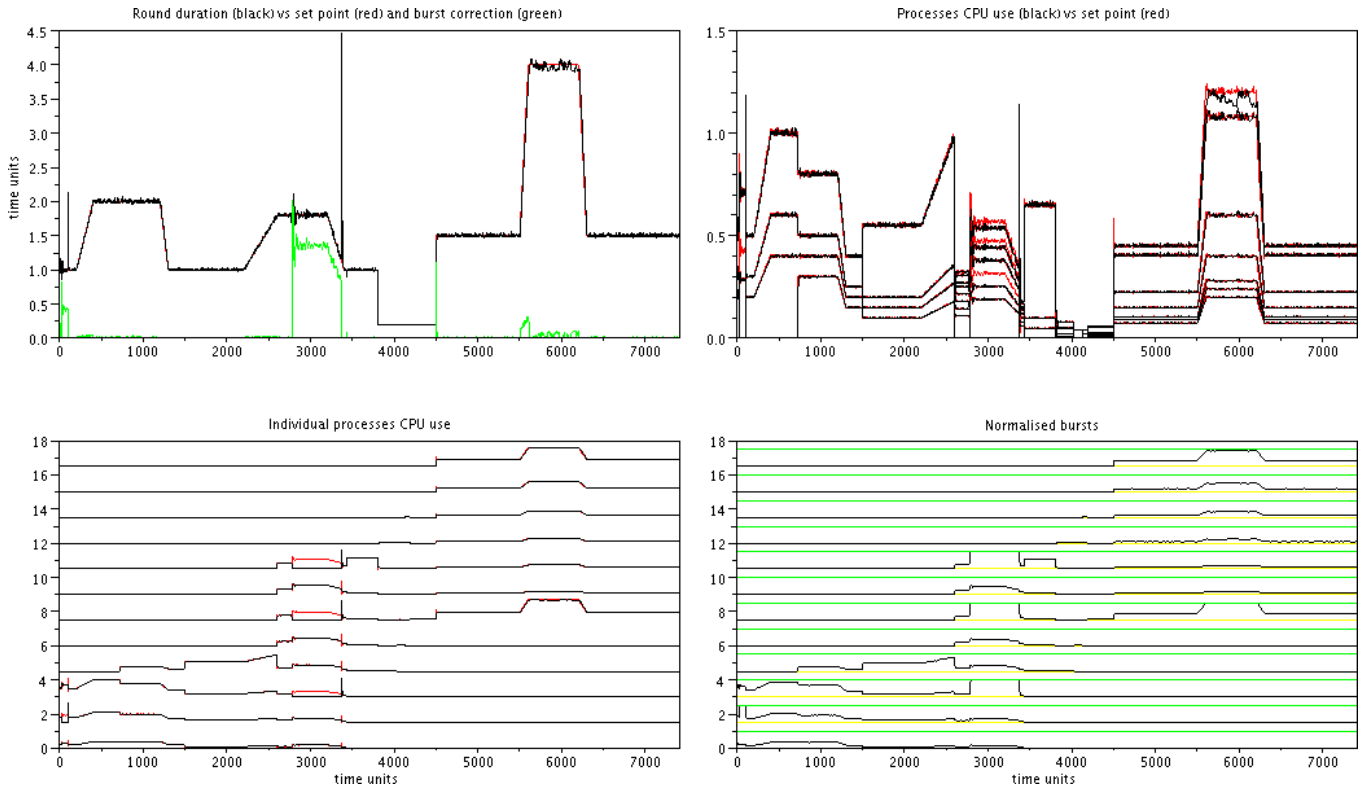

Fig. 5. Example of the results achievable with the RHPC in the presence of process blockings.

tically never pose state accessibility problems. Also those research developments, for which the present work is *de facto* an important preparatory stage, will be documented in the future.

REFERENCES

L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analysis of a reservation-based feedback scheduler. In *Proc. 23rd IEEE Real-Time Systems Symposium, RTSS 2002.*, pages 71–80, 2002.

M. Bayan and J. Cangussu. Automatic feedback, control-based, stress and load testing. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 661–666, New York, NY, USA, 2008. ACM.

E.F. Camacho and C. Bordons. *Model predictive control*. Springer Verlag, 2004.

S.L. Campbell, J.P. Chancelier, and R. Nikoukhah. *Modeling and simulation in Scilab/Scicos*. Springer Science+Business Media, New York, NY, USA, 2006. ISBN 0-387-27802-8 (hardcover).

R. Dantu, J.W. Cangussu, and S. Patwardhan. Fast worm containment using feedback control. *IEEE Transactions on Dependable and Secure Computing*, 4(2):119–136, April 2007.

J. Hellerstein, Y. Diao, S. Parekh, and D.M. Tilbury. *Feedback control of computing systems*. Wiley, 2004.

J.L. Hellerstein, V. Morrison, and E. Eilebrecht. Applying control theory in the real world: experience with building a controller for the .net thread pool. *SIGMETRICS Performance Evaluation Review*, 37(3):38–42, 2009.

O.P. Kreidl and T.M. Frazier. Feedback control applied to survivability: a host-based autonomic defense system. *IEEE Transactions on Reliability*, 53(1):148–166, March 2004.

A. Leva and M. Maggio. Feedback process scheduling with simple discrete-time control structures. *IET Proceedings - Control Theory and Applications*, 2010. (to appear).

C. Lu, J.A. Stankovic, G. Tao, and S.H. Son. Design and evaluation of a feedback control edf scheduling algorithm. In *Proc. 20th IEEE Real-Time Systems Symposium*, pages 56–67, Phoenix, AZ, USA, 1999.

C. Lu, J.A. Stankovic, and S.H. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Journal of Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, 23:85–126, 2002.

M. Maggio and A. Leva. A new perspective proposal for preemptive feedback scheduling. *International Journal of Innovative Computing, Information and Control*, 6(10), 2010.

M. Maggio, F. Terraneo, and A. Leva. Implementation and evaluation of a control-theoretical scheduler. In *Proc. 3rd International Symposium on Intelligent Informatics*, 2010.

L. Palopoli, L. Abeni, and G. Lipari. On the application of hybrid control to CPU reservations. In *Proc. Hybrid Systems Computation and Control*, 2003.

F. Xia and Y. Sun. Control-scheduling codesign: A perspective on integrating control and computing. *Dynamics of Continuous, Discrete and Impulsive Systems*, 13(1):1352–1358, 2006.