PID-based controls in computing systems: a brief survey and some research directions

Alberto Leva

Dipartimento di Elettronica e Informazione e Bioingegneria, Politecnico di Milano, Italy (e-mail: alberto.leva@polimi.it).

Abstract Applying controls to manage and optimise the behaviour of computers and networks is an important research field. In recent years, controllers have been gaining a role not only as add-ons to improve the efficiency of already functioning systems, but also as core components of those system themselves, and of their design. This paper provides a brief but reasoned review on the matter, evidencing the preminent role of PID-centred control solutions, and outlines some open issues for future research directions.

Keywords: Computing systems control; PID-based controls; Control-based system design.

1. INTRODUCTION

The idea of using control to improve the operation of computing system dates back to more than 15 years ago (Abdelzaher et al., 2003; Hellerstein et al., 2004). At that time, the main goal was to automatically (in computer jargon, *dynamically*) adapt some parameters of a system – for example, priorities in a scheduler – so as to improve the system operation. Used like this, control theory is a means to formalise "improvements" (e.g., in terms of cost functions to minimise) and to formally assess solutions—two important advantages with respect to the main alternative, i.e., mere heuristics. As such, the interest for "computing systems control" is high in both the academic and the information technology communities.

More recently, the idea was proposed to bring control much deeper inside computers and networks—i.e., for example, to design core components of operating systems entirely as controllers (Leva et al., 2013). This research has given rise to many applications as well, and despite it is still in its infancy, the efficiency improvement it appears to yield are already encountering technological interest.

It is impossible to provide here a complete description of so complex a *scenario*. On the contrary, however, it is quite easy to notice that PI/PID control, and other very similar techniques, play an important role in this story. Based on a decade of experience in the field, it is therefore interesting in the first place to ask oneself why, and then to reason on possible research directions, in the field of PID control and its vicinity, triggered and guided by its use in the context of computing systems. This is the purpose of this paper.

2. MINIMAL LITERATURE REVIEW

The outset of "computing systems control" is very well described in the paper that Diao et al. (2005) wrote as a follow-up to an IBM research report (RC23646, W0506-124, 29 June 2005). According to the authors, research needs to explore "the extent to which control theory can provide an architectural and analytic foundation for building self-managing systems" and to exploit the "correspondence between the elements of autonomic systems and those in control systems", as "control theory provides a rich set of methodologies for building automated self-diagnosis and self-repair systems with properties such as stability, short settling times, and accurate regulation".

A similar attitude is observed in Part I of the book by Hellerstein et al. (2004). To mention just a very few of the numerous works originated by those pioneering ideas, Lu et al. (2002) added a control layer to the EDF (Earliest Deadline First) and the rate monotonic scheduling algorithms, Xu et al. (2006) employed input-output models for resource allocation in data centres, Wang et al. (2007) proposed a control scheme for real-time systems utilisation, Kihl et al. (2008) analysed admission control in web servers by using a dynamic model, and so forth; the survey by Huebscher and McCann (2008) gives a comprehensive view of research in that period.

More recently, two additional trends emerged, and are gaining relevance. First, works like (Janert, 2013) stress the application-related usefulness of "control in computers" stronger than before. Second, the research focus is widening toward control-based computing systems *design* (Leva et al., 2013) as a means to avoid the problems of previous approaches, where *first* the computing system is designed to be completely functional, and only *afterwards* does control move in to "optimise" – whatever this means – its operation. Bringing control into the design of systems is a very articulated task, however, requiring to address multiple objectives. It is often necessary to *instrument* the system to allow closing the required loops (Brun et al., 2009; Patikirikorala et al., 2012), to design and install specific control-based components (Hoffmann et al., 2013; Arcelli et al., 2015), and sometimes to completely re-design (in a control-centric manner) parts that otherwise would be too critical for the desired results (Terraneo et al., 2014; Al-Areqi et al., 2015).

Summarising, a research topic of great importance is nowadays not only to control computing systems, but also

to make their design *control-aware*, and set up controls accordingly. In fact, in computing systems, most of the objects to be controlled are homogeneous to controllers, i.e., they are software themselves; and in such an *arena*, quite intuitively, the idea of co-design should find an application domain of election.

Given the breadth of the domain, as sketched out above, this research is expected to last for quite a long time. Nonetheless, and somehow despite the variety of encountered cases, there are conceptually grounded reasons to envisage a relevant role for PI- and PID-based controls. As a consequence, there are problem-specific control structures to build and assess, domain-specific stability and performance problems to study, and implementation-related facts – particularly, but not exclusively, measurement and actuation ones – to address in a systematic manner, so as to establish best practices and system design guidelines.

Viewed in general, the problem is to realise a cultural and engineering convergence for the computers and the control communities, but such a subject apparently strays form the scope of this paper. Limiting the scope to the present and expected role of PI/PID control, however, some interesting considerations can be made, and this is the purpose of the following sections.

3. PI(D) CONTROL IN COMPUTING SYSTEMS

When presenting his keynote paper, Hägglund (2012) suggested that in modern systems, PID blocks relate themselves to the overall control application like ants to their colony. This suggestive view can be re-formulated, for the scope of this work, by saying that such blocks act locally and near to the physics of the controlled system, while the effects of their operation becomes evident, and is evaluated, at higher levels in the system.

The reader may object that this is true in any control hierarchy, hence not specific to computing systems. True in principle, but with some relevant peculiarities. For our purposes, we just mention and briefly discuss two.

First, in any other domain but computing, there is a physically defined level below which neither measurements can be feasibly taken, nor actions are possible. To explain with a deliberately extreme example, a bit brutalised for brevity, temperature is physically governed by molecular motion, but one simply cannot think of measuring and actuating at that scale. In computing systems, the same is not so true. For example, SLAs (Service level Agreements) for a server are invariantly established on an average basis, such as a waiting time below 100ms for 99% of the submitted requests; however, not only one *can* think of acting at the level of "molecules" – sticking to the same case, the individual requests – but in several situations, this is exactly what one *must* do.

Second, outside the computing domain, lower hierarchy levels correspond to less arbitrary choices concerning measurement and actuation. For example, in a "peripheral" pressure loop the choice of sensor and actuator is mostly a matter of available/applicable/preferable technology, and there are well established guidelines to address it; on the contrary, "central" controls may have to do with "product quality" or other management-oriented performance indicators, which need *computing* from measurements, and that can be defined and obtained in conceptually different manners. Here too, in computing systems the *panorama* is often not equally clear. For example, measuring the throughput of a server is heavily influenced by the reference time interval adopted, and performance indices like the residence time of a queue need discussing right from their definition and interpretation if – as is frequently the case – input and output rates vary continuously, so that there is simply no such thing as a steady state.

Besides additional difficulties with respect to other domains, fortunately, the computing one also brings some good news. The main one, on which in this work we concentrate, is that the deeper one digs into the system, the simpler the dynamic equations describing the observed phenomena normally tend to become. Indeed, remarkable benefits come from endowing systems with a hierarchically "low" layer of PI- or PID-based controls targeted at mitigating the effects of exogenous disturbances on systems with quite simple dynamics. The following section provides some examples of this idea, showing that several heterogeneous problems, when viewed the way sketched above, reveal a uniform structure, and that PI- or PID-based control structures are very well suited to address them.

4. ONE MODEL FOR MANY CONTROLS

Consider the system described in the discrete time by

$$y(k) = y(k-1) + \mu\left(g(k-1)u(k-1) + d(k-1)\right) \quad (1)$$

where $u(k)$ is the control input, $y(k)$ the controlled variable, $g(k)$ a multiplicative disturbance (or equivalently, a time-varying gain), and $d(k)$ an additive load one. Since $g(k)$ is frequently interpreted as a variable "efficiency", as illustrated below, we assume

$$0 < g_{min} \leq g(k) \leq 1 \qquad \forall k. \quad (2)$$

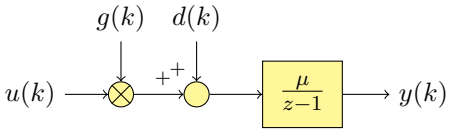

Figure 1. Delayed integrator with varying gain and additive load disturbance.

The system (1), shown as block diagram in Figure 1, fits a quite surprising variety of dynamics seen by PI(D) controllers as per the *scenario* described in Section 3. Let us see some examples.

(1) *In uniprocessor preemptive scheduling* the processor time received by a task at the time of the k-th intervention of the scheduler ($y(k)$) equals that before the said intervention ($y(k-1)$) plus the time allotted to the task at that intervention ($u(k-1)$) plus the effect of any phenomenon ($d(k-1)$) causing the actually used time to differ from the allotted one; in this case g and μ are structurally unitary.
(2) *In batch data processing* the amount of data processed by a task at the time of the k-th intervention of the resource allocator ($y(k)$) equals that before the said intervention ($y(k-1)$) plus the data nominally processed in the time between the two interventions by

the allotted computational resources. These resources are $u(k-1)$, and the nominal resource-to-processing-speed gain μ is multiplied by the time-varying "efficiency" $g(k-1)$, that accounts, among other effects, for the fact that different data may stress the allotted resources in an *a priori* unpredictably different manner; in this case d is structurally zero.

(3) *In queue-based services* the length of a queue ($y(k)$) at the time of the k-th intervention of the manager allotting computational power to the corresponding server equals that before the said intervention – i.e., $y(k-1)$) – plus the requests arrived in between the two interventions ($d(k-1)$) minus the requests processed in the same time span, i.e., the computational power allotted at the beginning of the span ($u(k-1)$) multiplied by a nominal gain μ, that represents the maximum server speed, and by the time-varying (or equivalently, data-dependent) efficiency $g(k-1)$.

(4) *In networks with multiple clocks and a time master* the time error between each clock and the master at the k-th synchronisation event equals the same error after the previous synchronisation event ($y(k-1)$) plus the integral $d(k)$ of the slave-to-master clock skew (inverse of the normalised frequency difference) over the time span between the two events minus a correction $u(k-1)$ computed at the previous event; in this case g and μ are structurally unitary.

As long as the control action is computed periodically – a hypothesis that can sometimes be relaxed but to which in this paper for simplicity we stick – the examples above should convince the reader that the equation "work accomplished now equals work accomplished at the last step plus allotted resource times efficiency plus additive effects", or convenient re-phrasings of it, indeed fit a vast number of physically heterogeneous applications. Adopting this paradigm, the differences among those applications reside in the presence or absence of the multiplicative and additive disturbance, in the way those disturbances can be characterised, and in the aspect of the reference signal to be tracked by the controlled variable. In any case, the aptitude of PI(D)-based controllers should be quite apparent.

To exemplify, when subjecting (1), that in state space form reads

$$\begin{cases} x_P(k) = x_P(k-1) + \mu\left(g(k-1)u(k-1) + d(k-1)\right) \\ y(k) = x_P(k) \end{cases} \tag{3}$$

to PI control, that for convenience we write as

$$\begin{cases} x_C(k) = x_C(k-1) + b_C e(k-1) \\ u(k) = x_C(k) + d_C e(k) \end{cases} \tag{4}$$

where $e(k) = w(k) - y(k)$ is the error, $w(k)$ being the reference signal, we get a closed-loop dynamics described by the LPV system with state $x(k) = [x_P(k)\ x_C(k)]'$ and dynamic matrix

$$A(k) = \begin{bmatrix} 1 - \mu g(k) d_C & \mu g(k) \\ -b_C & 1 \end{bmatrix} \tag{5}$$

In force of Theorem 1 by Akar and Narendra (2001), the origin of the state space is a globally asymptotically stable equilibrium for the closed-loop system – in the linear context wehere stability analysis is most typically carried out – iff the matrix pencils

$$\begin{aligned} H(\alpha, A_1, A_2) &= (0.5I - G(\alpha, A_1, A_2))^{-1} \\ &\quad (0.5I + G(\alpha, A_1, A_2)), \\ G(\alpha, A_1, A_2) &= \alpha P + (1-\alpha) Q \end{aligned} \tag{6}$$

where $\alpha \in [0,1]$, $P = 0.5I - (I + A_1)^{-1}$, $Q = 0.5I - (I + A_2)^{-1}$, and

$$A_1 = \begin{bmatrix} 1 - \mu g_{min} d_C & \mu g_{min} \\ -b_C & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 1 - \mu d_C & \mu \\ -b_C & 1 \end{bmatrix} \tag{7}$$

are Schur. In turn, by applying the conditions set forth in (Akar and Narendra, 2001, Section 4) it can be proven – computations are long and off-topic here, thus omitted for space limitations – that if g_{min} is taken arbitrarily close to zero, which is reasonable and even conservative, then the pencils (6) are Schur iff so are A_1 and A_2.

Hence, with a simple model covering a variety of control cases, stability (in the sense above) is straightforward to assess also in the presence of both multiplicative and additive disturbances. Also, given the simplicity of the stability condition, virtually any PI tuning rule for integrating processes can be safely used, and this is why in the following we do not focus on tuning to make room for more interesting domain-specific considerations.

We now illustrate the statement that "a simple model fits a variety of cases" a bit more in detail. The scheduling case, giving rise to a PI-based cascade structure, has already been treated in (Maggio et al., 2012), to which for brevity the reader is referred as the problem description therein – albeit at that time preliminary – fits very well the statements made here. As such, we concentrate on the other cases. Also, to answer in advance the objection – frequently encountered in the computer community – that "our models are too simple to represent reality well enough to carry out any design", we notice that all the controls here briefly presented were tested on high-fidelity simulators, and quite often also on the real systems, yielding results in very good accordance with the model forecasts. The provided references contain details.

4.1 Batch data processing

Processing a batch of data with unknown characteristics within a given deadline is an elementary operation necessary for the correct operation of many applications, for example "big data" ones. The challenge is to automatically absorb data-originated processing rate variability by allotting computational power, while avoiding over-provisioning to not unduly make other applications fail.

Here the set point can be viewed as a ramp (the desired completion), while the disturbance takes the form of small fluctuations – the high data rate tends to average out record-to-record variability – more or less of Brownian type, superimposed to large, abrupt but sporadic variations. These in turn occur at two main time scales: one corresponds to the clock frequency modifications produced by the power/performance governor (say every 100ms or so, a very long time with respect to processing the typical record), the other – even longer – is related to exogenous events like the failure of one or more processors.

Accepting short-term disturbances to partially pass on to the controlled variable, which is reasonable given the

inherent smoothing mechanism just quoted, the loop has to contain two integrators so as to keep the desired processing *rate*. PI control is therefore suitable.

At present, after experimenting with (real) cloud applications where resources can be rapidly allotted via the use of Docker containers, work is in progress to extend a big data framework (Spark) with the control functionalities just described.

4.2 Queue-based services

If the length ℓ of a queue and the inlet rate (requests over time) r_i are constant, then the residence (or waiting) time for a request in the queue is ℓ/r_i, constant as well. If – as in any real-world case – the inlet request rate varies, things become questionable. However, as discussed in Arcelli et al. (2016), queuing networks do benefit from having each server endowed with a local controller capable of maintaining the queue length at a desired level. The queue length set point may come for example from a higher-level response time controller. The advantage with respect to just processing requests as they arrive, is that the usage of computational resources, with PI loops in place at servers, is significantly less bursty and random, which simplifies the allocation problem at the level of the overall network.

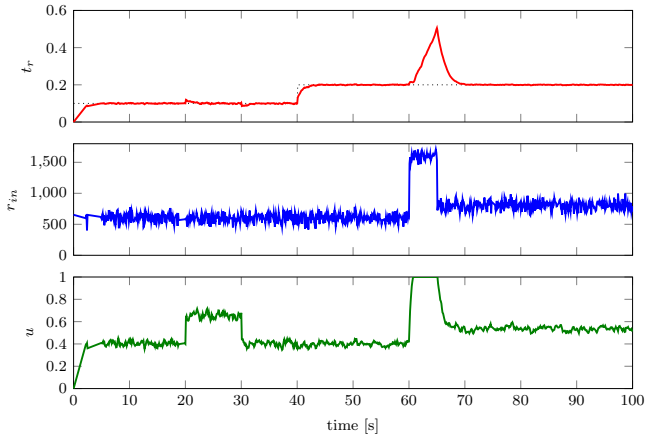

Figure 2. Queuing node control – simulation experiment.

Figure 2 reports a simulation experiment to give an idea of the achievable control quality. The top plot shows the response time, closely following the reference unless when the variable input rate (centre) is so high to cause the server command u (bottom, normalised in the [0,1] range) to be saturated. Note however the prompt recovery, as well as the rapid absorption of a 40% server efficiency drop between 20 and 30 seconds (barely visible on the controlled variable, well evident on the control signal). For full details on this problem and the relative control synthesis, the reader can refer to Arcelli et al. (2016).

4.3 Networks with multiple clocks

When synchronising a slave clock to a master that transmits a "sync" packet with fixed period T, the additive disturbance d is the accumulated slave clock skew over T, while the control signal u is naturally chosen to be an additive correction to the slave time, attempting to match $T + u$ in that time to T in the master one (see Leva et al. (2016) for a complete discussion). Hence, denoting by e the error, in this case (1) reduces to

$$e(k) = e(k-1) + u(k-1) + d(k-1), \tag{8}$$

i.e., in transfer function form,

$$E(z) = P(z)\left(U(z) + D(z)\right), \qquad P(z) = \frac{1}{z-1}. \tag{9}$$

In this case, the main source of disturbance is the temperature of the slave, that typically follows exponential transients with durations such that, over some synchronisation period, d significantly resembles a ramp. A PI is therefore not enough. To effectively reject the expected disturbance, it is convenient to prescribe the disturbance-to-output transfer function to have two unity zeroes, thus obtaining the controller $C(z)$ from

$$\frac{P(z)}{1 + C(z)P(z)} = \frac{(z-1)^2}{(z-\alpha)^3} \tag{10}$$

where α is a design parameter to trade error convergence speed versus high-frequency control sensitivity. This yields

$$C(z) = \frac{3(1-\alpha)z^2 - 3(1-\alpha^2)z + 1 - \alpha^3}{(z-1)^2}, \tag{11}$$

that can be interpreted as an ideal PID plus an additional integrator, or the cascade of two PIs. In this case the system is time-invariant, however, so that ensuring stability is a no-problem.

The so obtained controller significantly outperforms state of the art alternatives like the Flooding Time Synchronisation Protocol (Maróti et al., 2004) or the Feedback Based Synchronisation scheme (Chen et al., 2010), as shown in the experiment of Figure 3, where three wireless nodes carrying the three techniques (the proposed one is named FLOPSYNC-2 for Feedback-based LOw-Power SYNChronisation version 2) were subjected to a shade-sunlight transition causing a 20°C temperature variation. Note the ramp-like aspect of the temperature disturbance over say 3–4 synchronisation periods: FTSP and FBS make the error settle when temperature settles, while FLOPSYNC-2, thanks to the disturbance-.tailored controller, acts in advance. For more details on the control synthesis in FLOPSYNC-2, the reader is referred to Leva et al. (2016).

5. GENERAL REMARKS AND OPEN PROBLEMS

We have briefly reviewed a few control problems (in about ten years of research others were encountered) that are relevant for the management and the design of computing systems. We can now make some general remarks.

First, all these problems can be viewed as the control of (dominantly) first-order processes – in our review we further limited the scope to pure delayed integrators – subjected to multiplicative and additive disturbances. This problem formulation was achieved by deliberately aiming at "modelling the core phenomenon" behind he behaviour of the system; quite frequently, adopting his approach, simple dynamic balance equations prove to be enough to describe the system—although at quite low a level with respect to the one where quality of service is typically judged.

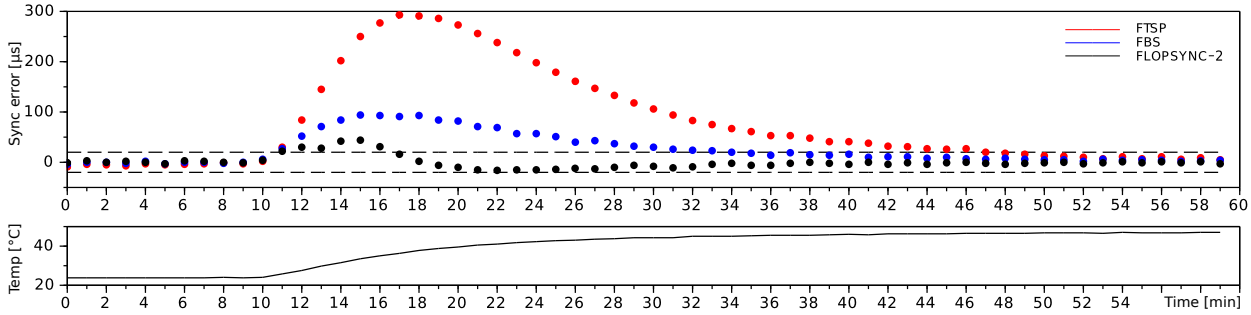

Figure 3. Synchronisation experiment: the top plot shows the error with FLOPSYNC-2 (black) versus FTSP (red) and FBS (blue) for the temperature transient in the bottom plot; the synchronisation period is 1min.

Then, as a consequence of this modelling attitude, uncertainty can very often be confined to *inputs* for the controlled system. In the case of linear models this is extremely desirable, because additive disturbances cannot influence stability. Also, given the low orders encountered, multiplicative ones can very often be treated straightforwardly. In such a *scenario*, PI- or PID-based controls prove very successful, either to tackle the problem entirely (as in uniprocessor scheduling) or as "peripheral" controls (like in queuing systems) to enforce local determinism and allow higher-level decision mechanisms – in the sense of hierarchical control – to reason in the face of a mitigated process variability.

Finally, the achieved control quality is in general satisfactory, and this is particularly good news because in computing systems more articulated – e.g., predictive – approaches easily fail, owing to the scarce possibilities of forecasting disturbances at the required time scale (think of guessing in advance when an unknown software will request resources, for example).

Besides these remarks, there is another point in favour of simple (e.g., PI) controls, that for unknown reasons – at least, to the best of the author's knowledge – is hardly ever mentioned in the literature. When somebody sells control applications, the processor time used to compute the control signals, is value. When somebody sells any other services, the processor time spent executing controllers to manage those services, is *stolen* to the service themselves. It is true that with modern computers, much more complex controls than PIs can be set up for problems like those shown above, but to justify the additional overhead, even modest, the control quality improvement has to be *really* significant. Indeed, it is quite likely that PI/PID-based control still has a bright future in this field.

Coming now to a couple of relevant open problems, although many schemes reduce to sets of single-loop SISO controllers, there are cases in which one needs to address a multivariable, interacting system. The peculiarity of the computing domain, in this respect, is that the great majority of couplings are generated by shared resources. In other words, there are several cases (scheduling is one) where control variables are not individually constrained, only their sum (or in general, a linear convex combination) is. Of course a conceptually viable way to address such a case is constrained MPC, but in the light of the efficiency considerations above, antiwindup-based solutions are preferable. There is thus the need for devising *ad hoc* solutions, possibly along "directional" approaches like that proposed by Horla (2009).

Moreover, any feedback control needs a sensor and an actuator, but in computing systems it may be difficult to measure the controlled variable, and even more to exert the control action, as sometimes these operations have to rely on operating system modules that were not conceived for control, and as a consequence – for example – tey do not provide the required precision, or the required timing guarantees, or any combination thereof.

Research is nowadays addressing these problems, for example "instrumenting" applications (Hoffmann et al., 2010). However this activity is mainly carried out in the computer science community, thereby not always paying due attention to facts (e.g., timing) that are conversely very important in system-theoretical terms, while the control community frequently just takes the available measurement and actuation machinery as a matter of fact. No criticism, but more convergence is needed.

6. CONCLUSIONS AND FUTURE WORK

We have reviewed some research results on control and control-based design of computing systems. The conclusion drawn, and the main point of this "reasoned retrospect" paper, is that not only PI- and PID-based controls play a relevant role, but in several heterogeneous applications they face basically the same dynamics. Differences between the said applications reside in the aspect of the reference signal and of disturbances. As such, one can easily envisage at least two main research directions for future work.

First, on the methodological side, there is in some cases the need for domain-specific tuning procedures – not necessary *rules*, beware – to fit the feasible experiments and to not disturb the system operation when a controller (re-)tuning is in progress. This is because, as said before, well assessed tuning rules normally do a perfect job, but in a system subject to highly variable requests on the part of many actors (users, processes, and so forth) it may be problematic to apply *stimuli* – think of steps or relay feedback – that are conversely the standard e.g. in process control. Addressing this aspect, together with a qualification for the aplicability limits of the simple models here discussed – would open the way to many other applications other than those shown herein, where the model is either parameter-free or a "worst case" parametrisation is quite easily obtained.

On the technological side, given the uniformity here evidenced, it would be desirable to have standard PI- or PID-based components to include in operating systems, and possibly in firmware. Control could then rely on them, to the advantage of compactness, performance, and maintainability. However, given the heterogeneous nature of systems to date, this is not so easy to implement.

Finally, as noticed above, present systems frequently exhibit highly layered software structures, making it difficult to reach sensing and actuating points. As such, if on one hand it is necessary to make control-based systems manageable by the standard administrator, who has no (and needs not have) a control culture, on the other hand there is an undoubted need to make systems more control-friendly. In the opinion of the author this is a great challenge, and addressing it successfully will lead to correspondingly greater performance in the future.

REFERENCES

Abdelzaher, T., Stankovic, J., Lu, C., Zhang, R., and Lu, Y. (2003). Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3), 74–90.

Akar, M. and Narendra, K. (2001). On the existence of a common quadratic Lyapunov function for two stable second order LTI discrete-time systems. In *Proc. 2001 American Control Conference*, 2572–2577. Arlington, VA, USA.

Al-Areqi, S., Görges, D., and Liu, S. (2015). Event-based networked control and scheduling codesign with guaranteed performance. *Automatica*, 57(7), 128–134.

Arcelli, D., Cortellessa, V., Filieri, A., and Leva, A. (2015). Control theory for model-based performance-driven software adaptation. In *Proc. 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, 11–20. New York, NY, USA.

Arcelli, D., Cortellessa, V., and Leva, A. (2016). A library of modeling components for adaptive queuing networks. In *Proc. 13th European Workshop on Performance Engineering*, 204–219. Chios.

Brun, Y., Di Marzo Serugendo, G., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Müller, H., Pezzè, M., and Shaw, M. (2009). Engineering self-adaptive systems through feedback loops. In B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee (eds.), *Software engineering for self-adaptive systems*, 48–70. Springer, Berlin, Germany.

Chen, J., Yu, Q., Zhang, Y., Chen, H., and Sun, Y. (2010). Feedback-based clock synchronization in wireless sensor networks: A control theoretic approach. *IEEE Transactions on Vehicular Technology*, 59(6), 2963–2973.

Diao, Y., Hellerstein, J., Parekh, S., Griffith, R., Kaiser, G., and Phung, D. (2005). A control theory foundation for self-managing computing systems. *IEEE journal on selected areas in communications*, 23(12), 2213–2222.

Hägglund, T. (2012). Signal filtering in PID control. In *Proc. 2nd IFAC Conference on Advances in PID Control*, 1–10. Brescia, Italy.

Hellerstein, J., Diao, Y., Parekh, S., and Tilbury, D. (2004). *Feedback control of computing systems*. John Wiley & Sons, New York, NY, USA.

Hoffmann, H., Eastep, J., Santambrogio, M., Miller, J., and Agarwal, A. (2010). Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *Proc. 7th international conference on Autonomic computing*, 79–88. Washington, DC, USA.

Hoffmann, H., Maggio, M., Santambrogio, M., Leva, A., and Agarwal, A. (2013). A generalized software framework for accurate and efficient management of performance goals. In *Proc. 2013 International Conference on Embedded Software*, Article No. 6658597. Montréal, Canada.

Horla, D. (2009). On directional change and anti-windup compensation in multivariable control systems. *International Journal of Applied Mathematics and Computer Science*, 19(2), 281–289.

Huebscher, M. and McCann, J. (2008). A survey of autonomic computing – degrees, models, and applications. *ACM Computing Surveys*, 40(3), 7:1–7:28.

Janert, P. (2013). *Feedback control for computer systems*. O'Reilly Media, Sebastopol, CA, USA.

Kihl, M., Robertsson, A., Andersson, A., and Wittenmark, B. (2008). Control-theoretic analysis of admission control mechanisms for web server systems. *World Wide Web*, 11(1), 93–116.

Leva, A., Maggio, M., Papadopoulos, A., and Terraneo, F. (2013). *Control-based operating system design*. IET, London, UK.

Leva, A., Terraneo, F., Rinaldi, L., Papadopoulos, A., and Maggio, M. (2016). High-precision low-power wireless nodes synchronization via decentralized control. *IEEE Transactions on Control Systems Technology*, 24(4), 1279–1293.

Lu, C., Stankovic, J., Son, S., and Tao, G. (2002). Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23, 85–126.

Maggio, M., Terraneo, F., Papadopoulos, A., and Leva, A. (2012). A PI-based control structure as an operating system scheduler. In *Proc. 2nd IFAC Conference on Advances in PID Control*, 329–334. Brescia, Italy.

Maróti, M., Kusy, B., Simon, G., and Lédeczi, A. (2004). The flooding time synchronization protocol. In *Proc. 2nd International Conference on Embedded Networked Sensor Systems*, 39–49. Baltimore, MD, USA.

Patikirikorala, T., Colman, A., Han, J., and Wang, L. (2012). A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Proc. 2012 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 33–42. Zürich, Switzerland.

Terraneo, F., Rinaldi, L., Maggio, M., Papadopoulos, A., and Leva, A. (2014). FLOPSYNC-2: sub-microsecond, sub-μa clock synchronisation for wireless sensor networks. In *Proc. IEEE Real-Time Systems Symposium RTSS 2014*, 11–20. Roma, Italy.

Wang, X., Jia, D., Lu, C., and Koutsoukos, X. (2007). Deucon: Decentralized end-to-end utilization control for distributed real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 18(7), 996–1009.

Xu, W., Zhu, X., Singhal, S., and Wang, Z. (2006). Predictive control for dynamic resource allocation in enterprise data centers. In *Proc. 10th IEEE Network Operations and Management Symposium*, 115–126. Vancouver, Canada.