Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 52-27 (2019) 38–43

An event-based multi-purpose approach to computational sprinting

Alberto Leva * **Federico Terraneo** * **Chiara Cimino** ** **Silvano Seva** ***

* *DEIB, Politecnico di Milano, Italy (e-mail: {alberto.leva,federico.terraneo}@polimi.it).*
** *PhD student at the DEIB (e-mail: chiara.cimino@polimi.it).*
*** *Graduate student at the DEIB (e-mail: silvano.seva@mail.polimi.it).*

Abstract Computational sprinting was introduced to tackle the "dark silicon" problem, i.e., to allow a processor to transiently consume a power that could not be sustained indefinitely without thermal damage. However, the idea of sprinting has other potential applications, also tightly related to embedded systems. In this paper we evidence a few of these, and discuss the *scenario* in a view to abstracting and defining general problems. We then propose an event-based approach to treat the said problems in a unitary manner.

Keywords: Event-based control, computational sprinting, performance boosting.

1. INTRODUCTION

In Raghavan et al. (2012), computational sprinting is defined as a technology by which "a chip temporarily exceeds its sustainable thermal power budget to provide instantaneous throughput, after which the chip must return to nominal operation to cool down". Given the unprecedented power densities of modern processors sprinting is deeply investigated, but mostly as a runtime means to trade performance versus thermal safety: a famous industrial realisation is the Intel Turbo Boost (Rotem et al., 2012).

Many systems – also embedded ones – operate in conditions and under requirements that make sprinting attractive; for example, a microcontroller in as harsh an environment as the engine compartment of a vehicle, could take profit of sprinting to reduce thermal stress.

In this paper we attempt to widen the perspective about sprinting, by addressing the research questions below.

- *RQ1.* Can the idea of "sprinting" be extended beyond that of transiently exceeding a power budget?
- *RQ2.* If so, is there a means to qualify the class of problems to which it can be applied?
- *RQ3.* If still so, can some unified approach to those problems be envisaged?

We then propose *event-based sprinting* as a design framework for a (qualitatively) qualified class of problems, addressing different purposes, related to the efficient and safe management of computing systems.

2. BRIEF LITERATURE REVIEW

To date, the literature on sprinting is centred on temporarily exceeding a thermal budget to deliver increased performance (Taylor, 2013). Foundations were laid down by works like Raghavan et al. (2012); Rotem et al. (2012); Raghavan et al. (2013), and developments are still being investigated, both methodologically (Fan et al., 2016) and technologically (Kondguli and Huang, 2018). There are also some modelling and simulation works (Lopez-Novoa et al., 2015), but with the computer engineering viewpoint, not the control-theoretical one adopted herein.

On the other hand, the idea of augmenting or diminishing the resources for an entity based on the state of that entity, the requirements it has to fulfil, and the condition of the system to which it belongs, has been envisaged in fields like quality of service (Ghosh et al., 2003), scheduling (Leva and Maggio, 2010; Maggio et al., 2014), containerized applications (Baresi et al., 2016), cloud computing (Zhang et al., 2011; Xiao et al., 2013), data centre efficiency (Zheng and Wang, 2015), communications (Suh and Mo, 2008; Čelik and Sung, 2018), infrastructure-as-a-service systems (Ataie et al., 2018), but not with a unitary methodological framework.

Several solutions were in fact derived based more on the available sensing and actuation machinery than on the dynamic character of the encountered control problems. A notable example is the number of papers that appeared once the Docker technology (Boettiger, 2015) became available to reduce actuation delays with respect to alternatives based on virtual machines (Seth and Singh, 2017), and make the said delays far more deterministic than before (Baresi et al., 2016; Guan et al., 2017). At the same time, methodological developments mostly concerned modelling and solution frameworks to reproduce the structure of the controlled systems, resorting e.g. to game-theoretic (Wei et al., 2010) or agent-based (Zahedi et al., 2017) approaches. The survey by Yousafzai et al. (2017)can provide the interested reader with a lot of additional references. Finally, as for the use of event-based control in this *arena*, despite its inherent keenness to embedded/distributed systems (Abdelaal et al., 2017), studies

2405-8963
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2019.12.729

are still confined to the mainstream sprinting framework, as witnessed e.g. by Chen et al. (2018).

Summing up, we believe that sprinting can be given a more general and unitary sense than the literature does to date. This paper attempts to provide a contribution by casting the problem into the framework of switching systems,for which a huge interest is testified by research and application works in diverse domains like control (Zhu and Antsaklis, 2015), computer engineering (Ma et al., 2017) and economics (Kim et al., 1999)—many of them sharing the idea of saving resources when possible, and use them to enhance performance when required.

3. GENERALISED COMPUTATIONAL SPRINTING

We introduce our general sprinting framework referring to the high-level description shown in Figure 1, that evidences its organisation into a time-based and an event-based part.

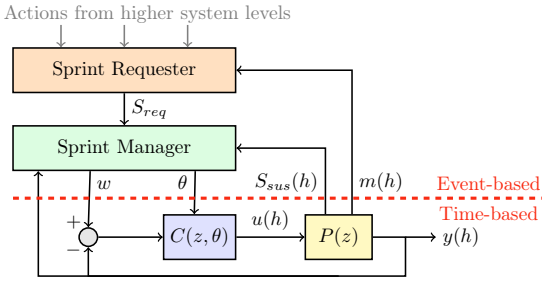

Figure 1. General scheme for the proposed event-based sprinting framework.

The time-based part is composed of the controller C and of the measurements m and S_{sus}, discussed later on, all executed periodically at step q and counted by the index h. The event-based part comprises the Sprint Requester (SR) and the Sprint Manager (SM) blocks. We assume the process to be LTI, with transfer function $P(z)$; $C(z,\theta)$ is conversely LPV, as the event-based part can change the set point w, but also modify the parameter vector θ.

3.1 The time-based part

We assume to date a SISO control loop. The controller is endowed with saturation management and antiwindup, hence θ contains the parameter of its transfer function, but also the upper and lower saturation values for the control signal u. The motivation for this will emerge shortly, when specialising the scheme to some cases of interest.

Given the above, the time-based part of the scheme is represented as

$$\begin{cases} x_P(h) = A_P x_P(h-1) + b_P u(h-1) \\ y(h) = c_P x_P(h) \\ e(h) = w(h) - y(h) \\ x_C(h) = A_C(\theta) x_C(h-1) + b_C(\theta) e(h-1) \\ u_{ns}(h) = c_C(\theta) x_C(h) + d_C(\theta) e(h) \\ u(h) = \max(u_{min}, \min(u_{max}, u_{ns}(h))) \end{cases} \quad (1)$$

where x_P and x_C are the state vectors of P and C, e is the error, the matrices of P are constant while those of C depend on θ, $[u_{min}, u_{max}]$ is the control signal range, and the other symbols have the meaning stemming from Figure 1. We do not spend further words on this part, if not for saying that in computing systems it is convenient to consider the actuator as part of the controller (an idea exploited later on).

3.2 The event-based part

We illustrate the event-based part by means of the automaton depicted in Figure 2.

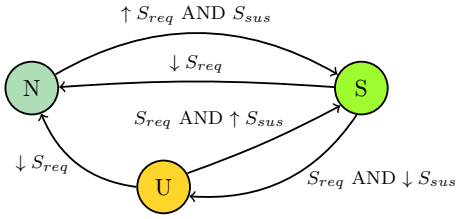

Figure 2. Transitioning among the (N)ormal, (S)print and (U)nder-performing operating modes.

The system has a "normal" (N) and a "sprint" (S) operating mode. The request or an N→S transition comes from the SR block in Figure 1 and is caused by exogenous actions denoted there as "from higher system levels", although it may account for some metrics m coming the time-based loop. The S→N transition can happen either due to an exogenous request as well, which means that remaining in S is not necessary anymore, or because the S mode is not further sustainable. The booleans S_{req} and S_{sus} dictate respectively that the S mode is requested and sustainable, while ↑ and ↓ indicate the false→true and the true→false transitions. If staying in S is not required the system goes back to N, while if S is not sustainable it reaches an "underperforming" (U) mode, from which it can go to N if the reasons to desire S cease, or back to S if the said reasons persist, and S becomes sustainable again.

4. SPRINTING PROBLEMS

To characterise our framework we map the structure just devised to some cases of interest, thus coming to a problem classification along the research questions of Section 1.

Starting with RQ1, one has a sprinting problem when there is a resource to use that is "normally" limited, but the limit can be sometimes superseded if (i) somebody asks for this, and (ii) no damage is caused. The application of such an idea to thermal management is immediate. However, if one takes a less physical viewpoint, a lot of resources need allotting, the reasons to ask for "more than usual" can be various, and the feasibility or infeasibility of that request may depend on a lot of equally various things.

In some cases S is not indefinitely sustainable owing to physical facts—thermal safety is the obvious example. In some others staying in S forever would physically be possible, but there may be economic reasons to not do so: for example, a server can have free virtual machines to allot to a customer, but doing this could result in over-provisioning, and depending on the stipulated cost plan, may or may not be profitable for the provider. Problems of this type are in fact optimal allocation ones, but computing (and especially embedded) systems require rapid decisions, and solving an optimisation problem online is

Problem	Reason(s) for S_{req}	Metric m	Condition S_{sus}	w and y	Controller (LPV)	Action of the S mode	Effect on θ and/or w
Processor thermal management	More performance needed	Perf. indic./ counters	CPU temperature acceptable	Temperature	Typically PI-like	Allow CPU to draw more power	Increase/decrease w
Power-aware resource allocation	All permitted units already allotted	Case-specific	Extra units available, power OK	Case-specific	Various types	Allow to allot more units	Increase/decrease upper saturation value for u
Application progress control	Allotted units used up to nearly 100%	Progress rate	Extra units in system not claimed	Progress rate	Integral or PI	Actuate on more units	Increase/decrease the C (i.e., the loop) gain
Network bandwidth distribution	Applications signal low data rate	Various, sometimes progress	Network bandwidth available	Various	Low-order, often purely proportional	Make data transfers faster	Increase/decrease actuation delay (modelled in C)
Adaptive queue networks	Occupations and/or service rates drift	Queue lengths, rates	Spare capacity available	Typically queue occupations	Low-order	Increase max server shares	Increase/decrease loop gain

Table 1. Synthetic description of some "generalised sprinting" problems.

often too demanding. In such cases, a sprinting-based approach like ours can be the right choice.

A sample of the results obtained by taking the general standpoint above, and mapping the proposed framework to real problems, is shown in Table 1, where we mention thermal management in the first place to better evidence the extension introduced with respect to the sprinting idea. As can be seen, the answer to RQ1 is evidently affirmative.

The general characteristics of a sprinting problem, as evidenced above, implicitly answer RQ2 as well. It is just worth adding that almost invariantly, entering the S mode is viewed by the control loop – where, recall, the actuator is considered part of the controller – as a modification of a set point, a saturation value, a gain, or a delay. It is now time to address RQ3, showing that a unitary approach can be found to address the class of problems just outlined.

5. AN EVENT-BASED UNIFIED APPROACH

Addressing systems described by (1) in the most general form would largely exceed both the available space and the scope of this paper, but on the other hand it would also be unduly complicated for many of the computing-related cases we target. We therefore start by introducing and justifying some simplifications.

5.1 Preliminaries

We assume $P(z)$ to have a low-order dynamics, asymptotically stable or integrating. This dynamics can sometimes be cascaded to a delay, that we assume to be an integer multiple of the timestep q for the time-based loop.

The first hypothesis is backed up by a number of cases covering all the domains quoted here and more, see e.g. the book Leva et al. (2013). The second is reasonable as long as event times are integer multiples of the quantum q; this is in turn justified because in a fully digital system like a computing one, all sensing events are clocked. In fact we are also assuming that the time- and the event-based parts of the system share the time quantum q and are synchronised to some clock, physical or virtual, with that period. This may be critical for distributed systems, but reasonable values for q are normally far higher than communication-induced delays, and even more important, than the *variability* of those delays: modern high-precision synchronisation techniques Terraneo et al. (2014) can mitigate delay-related issues, providing technologically sound solutions. Furthermore, we assume $C(z, \theta)$ to be low-order as well. This is justified by the wide use of integral or PI controllers in our applications, see again Leva et al. (2013) for a list of cases.

We finally have to assume that the closed time-based loop is asymptotically stable for any θ, as this is necessary for treating switching stability as we are going to do. In the context of this work, this leads to also conclude that some events (namely, modifications in the set point or in a saturation value) can affect performance but not stability.

5.2 Stability

Denoting by A_N and A_S the closed-loop dynamic matrices of the time-based loop in the N and S mode, we can state the following result.

Proposition 1. If both A_N and A_S are Schur and diagonalisable, two finite dwell times Δ_N and Δ_S can be computed that ensure asymptotic switching stability.

Proof. Indicating with ν_i and σ_i the eigenvalues of A_N and A_S, by hypothesis there exist two nonsingular matrices T_N and T_S such that

$$T_N^{-1} A_N T_N = diag\{\nu_i\}, \quad T_S^{-1} A_S T_S = diag\{\sigma_i\}. \tag{2}$$

hence the h-steps state transition matrix for the S mode (the N case is analogous) is

$$\Phi_S(h) = T_S \, diag\{\sigma_i^h\} \, T_S^{-1}. \tag{3}$$

Now consider a generic matrix norm $\| \cdot \|_p$ induced by the vector p-norm

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \tag{4}$$

in the state space. Since such a norm is an operator norm and therefore inherently submultiplicative, an overbound for $\|\Phi_S(h)\|_p$ is readily obtained in the form

$$\|\Phi_S(h)\|_p \leq c_p(T_S) \|diag\{\sigma_i^h\}\|_p, \tag{5}$$

where

$$c_p(T_S) = \|T_S\|_p \, \|T_S^{-1}\|_p \tag{6}$$

is the condition number for matrix T_S in norm p. Also, since by hypothesis $|\sigma_i| < 1 \ \forall i$, the overbound just

calculated monotonically decreases with h and tends to zero for $h \to \infty$, independently of p. Hence there surely exists an integer $\Delta_S \geq 0$ such that

$$\|\Phi_S(\Delta_S)\|_p < 1, \tag{7}$$

i.e., such that the system contracts the state norm after Δ_S steps in S mode; Δ_S is the sought S-mode dwell time. As said, the N case is analogous. □

Remark 1. The computed $\Delta_{N,S}$ do ensure switching stability as required, but are in fact over-estimates of the real minimum dwell times. Obvious, but worth noticing.

Remark 2. In the light of the remark above, to avoid unduly over-estimating the dwell times, among the infinite $T_{N,S}$ matrices one has to select those with the minimum condition number.

5.3 Realisability, performance, constraints

Having established a sufficient stability condition simple enough for use in an iterative design process, the question arises whether enforcing the estimated dwell times can impact the control system's operation to a critical extent.

In this respect, the two dwell times above may have different meanings. Precisely, if Δ_N is "too large" the system may not go into the S mode immediately when required, i.e., not "sprint" as much as possible, to the detriment of performance. If the "too large" dwell time is Δ_S, the system may be obliged to defer exiting the S mode, hence not release resources timely, hence impacting somebody else's performance, but even not return promptly enough to a safe operation, jeopardising its own health.

As such, for the scheme to be realisable and enhance performance via sprinting as much as possible, Δ_N should be minimised subject to a strict upper bound for Δ_S.

6. APPLICATION EXAMPLES

The problem of choosing the diagonalising matrices with minimum condition number, which in general is difficult, can however be quite easily treated in the second-order case, that luckily fits several situations of engineering interest, hence on which we concentrate.

6.1 Example 1

One of the situation just mentioned is application progress control Leva et al. (2018), which amounts to analysing a control loop with an integrating process and a PI controller, described respectively as

$$\begin{cases} x_P(k) = x_P(k-1) + b_P u(k-1) \\ y(k) = x_P(k) \end{cases} \tag{8}$$

and

$$\begin{cases} x_C(k) = x_C(k-1) + b_C\,(w(k-1) - y(k-1)) \\ u(k) = x_C(k) + d_C\,(w(k) - y(k)) \end{cases} \tag{9}$$

The interpretation of (8,9) is that u is a resource – typically, computational capability – and y the work accomplished by a software application, for example the number of processed "items" no matter what these are.

Physically, the N → S transition can take several forms. It can just be increasing the maximum allowed u, for example allowing a higher CPU share and/or the use of more CPUs, and in this case stability cannot be affected. It can however also be increasing the value of b_P, for example *replacing* a CPU with a more powerful one—a quite typical manoeuvre in the so called "big.LITTLE" architectures, frequently encountered in embedded systems. In this second case, we need to bring the analysis in Section 5.2 into play.

To lighten the notation, since the same considerations apply to the N and the S mode, in the following we denote the closed-loop dynamic matrix by just A and the estimated dwell time by Δ. This said, from (8,9) we have

$$A = \begin{bmatrix} 1 - b_P d_C & b_P \\ -b_C & 1 \end{bmatrix}. \tag{10}$$

The eigenvalues of A are assigned to be (λ_1, λ_2) by setting

$$b_C = \frac{(1-\lambda_1)(1-\lambda_2)}{b_P}, \quad d_C = \frac{2-\lambda_1-\lambda_2}{b_P}. \tag{11}$$

and the corresponding family of diagonalising matrices, with real parameter α, reads

$$T(\alpha) = \begin{bmatrix} 1 & \alpha \\ \frac{1-\lambda_1}{b_P} & \alpha\,\frac{1-\lambda_2}{b_P} \end{bmatrix}. \tag{12}$$

The condition number of $T(\alpha)$ in any p-norm does not depend on the sign of α, hence without loss of generality we assume $\alpha > 0$. We also intuitively suppose that increasing a resource speeds up an application, hence $b_P > 0$. Finally, to avoid oscillatory behaviours, we assume $\lambda_{1,2}$ to be real and lie in the [0,1) range.

All this said, the computational difficulty of enforcing (7) to find Δ, significantly depends on the chosen norm. For example, if one selects the ∞-norm, with the assumptions above the condition number of $T(\alpha)$ turns out to be

$$\begin{aligned} c_\infty(T(\alpha)) = {} & \max\left(\alpha + 1, \tfrac{1-\lambda_1+\alpha(1-\lambda_2)}{b_P}\right) \\ & \times \max\left(\tfrac{1+b_P-\lambda_1}{\alpha(\lambda_1-\lambda_2)}, \tfrac{1+b_P-\lambda_2}{\lambda_1-\lambda_2}\right), \end{aligned} \tag{13}$$

and defining

$$\lambda_{max} = \max(\lambda_1, \lambda_2), \tag{14}$$

the required dwell time is estimated as

$$\Delta_\infty = \min_\alpha \left\lceil \frac{\log(c_\infty(T(\alpha)))}{\log(1/\lambda_{max})} \right\rceil. \tag{15}$$

This is a simple procedure, but has two main drawbacks. First, the derivative of $c_\infty(T(\alpha))$ with α is not continuous, making it difficult to find the minimum c_∞ if not numerically. Second, one would prefer to use the 2-norm, as this is more easily interpreted than the ∞-norm, as $\|x\|_2 \geq \|x\|_\infty$ so that the 2-norm yields tighter convergence conditions, as the 2-norm is easy to relate to Lyapunov functions, and for other reasons inessential herein. However, the 2-norm (hence the corresponding condition number c_2) is well known to be harder to compute. To overcome this difficulty, we propose to go through the Frobenius condition number c_F, because for any square matrix M

$$c_F(M) = \|M\|_F \|M^{-1}\|_F \geq \|M\|_2 \|M^{-1}\|_2 = c_2(M), \tag{16}$$

thus estimating the 2-norm dwell time with $c_F(T(\alpha))$ in the place of the desired $c_2(T(\alpha))$, is conservative. In the case at hand, omitting trivial computations, we have

$$c_F(T(\alpha)) = \frac{b_P^2 + (1-\lambda_1)^2 + (b_P^2 + (1-\lambda_2)^2)\alpha^2}{b_P|\lambda_2 - \lambda_1|\alpha}, \tag{17}$$

whence the conservative estimate of Δ in the 2-norm

$$\widehat{\Delta}_2 = \min_{\alpha} \left\lceil \frac{\log(c_F(T(\alpha)))}{\log(1/\lambda_{max})} \right\rceil . \tag{18}$$

Taking the further conventional assumption $\lambda_1 > \lambda_2$, the minimum of Δ_2 as a real function of α occurs at

$$\alpha^o = \sqrt{\frac{b_P^2 + (1-\lambda_1)^2}{b_P^2 + (1-\lambda_2)^2}} \tag{19}$$

and this produces

$$\widehat{\Delta}_2 = \left\lceil \frac{\log\left(\frac{2\sqrt{b_P^2+(1-\lambda_1)^2}\sqrt{b_P^2+(1-\lambda_2)^2}}{b_P(\lambda_1-\lambda_2)}\right)}{\log(1/\lambda_1)} \right\rceil . \tag{20}$$

In general, the value of α that minimises the condition number in a certain norm, does not have the same effect in another norm; finding a counterexample is easy. It is however interesting to observe that for a 2×2 matrix, the above is instead true for the 2-norm and the Frobenius norm, i.e., α^o as per (19) minimises both. The proof is readily obtained by construction and is omitted for brevity, but further corroborates the *rationale* behind (20).

6.2 Example 2

A second case – actually a specialisation of the one just treated – is when λ_2 is set to zero, which is a quite common choice in progress control, as it guarantees tracking of a ramp set point (one integrator is in the process and one in the PI) with a response that could be called, stretching the terminology a bit, a first-order filtered deadbeat one.

In this case $\widehat{\Delta}_2$ is a function of two parameters only, namely b_P and λ_1, as (20) reduces to

$$\widehat{\Delta}_2 = \left\lceil \frac{\log\left(\frac{2\sqrt{b_P^2+1}\sqrt{b_P^2+(1-\lambda_1)^2}}{b_P\lambda_1}\right)}{\log(1/\lambda_1)} \right\rceil , \tag{21}$$

A plot of $\widehat{\Delta}_2$ as per (21) is shown in Figure 3. It can be observed that for any b_P, values of λ_1 closer to the unity result in larger dwell times. This is physically consistent: supposing that the control system ultimately governs some continuous-time phenomenon, and that the characteristics of that phenomenon dictate the dwell time, λ_1 closer to one means smaller sampling times, hence larger values of the dwell time when counted in samples.

Along the same reasoning, when starting from a continuous-time control synthesis, one should select the sampling time q for the time-based part of the control system – see the scheme in Figure 1 – so as to obtain a discrete-time pole λ_1 not exceeding say 0.6–0.8, and in this case dwell times below 4 are well achievable.

6.3 Example 3

Still on the same situation, we can examine the effect of an increment of b_P when transitioning from the N to the S mode. For simplicity we assume that in the N mode $b_P = 1$, while in the S mode $b_P = b_{PS} > 1$.

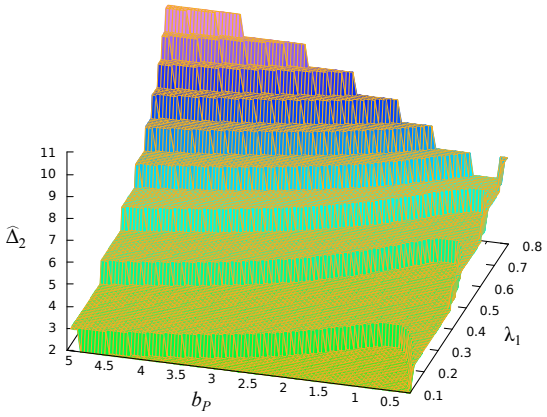

Figure 3. Plot of $\widehat{\Delta}_2$ as a function of (b_P, λ_1) with $\lambda_2 = 0$.

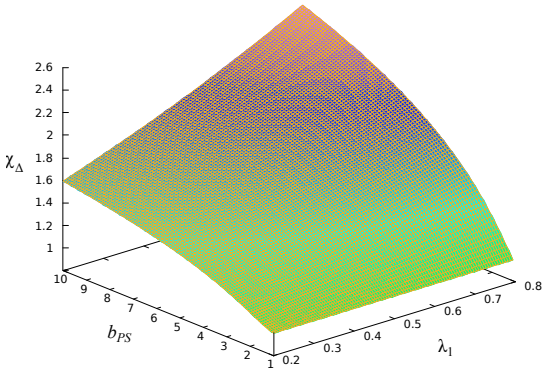

Figure 4. Relative dwell time increment for a multiplicative variation of b_P.

Defining χ_Δ as the ratio between $\widehat{\Delta}_2$ in the S and the N modes, Figure 4 summarises the effect of the corresponding process gain increment. As can be seen, multiplying b_P by a factor up to 10, at most triples the sampling time. However, such a factor is hardly conceivable in practice: for example, in progress control, S-mode speedups of 2–3 times is what one can normally expect because the spare resources required for higher accelerations are seldom (if ever) available in a well sized system. If this is the case and/or λ_1 is kept low, one can keep $\lceil \chi_\Delta \rceil$ below two.

7. CONCLUSIONS AND FUTURE WORK

We proposed a common framework to address computational sprinting applications, a subject of undue interest for the time- and energy-efficient operation of several modern computing systems.

We showed that different problems can be viewed in a unitary manner, and that event-based control is a natural framework to cast them. We proposed a methodology to analyse a sprinting application as a mixed time/event-based system, for stability and (preliminarily) for performance. We presented a few results to show that the proposed analysis method is sensible, and can effectively complement control design ones. Future work will be directed at treating the open issues mentioned in the paper, together with addressing some practical applications in computing systems control, and control-based design.

REFERENCES

Abdelaal, A., Hegazy, T., and Hefeeda, M. (2017). Event-based control as a cloud service. In *Proc. 2017 American Control Conference*, 1017–1023. Seattle, WA, USA.

Ataie, E., Entezari-Maleki, R., Ehsan, S.E., Egger, B., Ardagna, D., and Movaghar, A. (2018). Power-aware performance analysis of self-adaptive resource management in IaaS clouds. *Future Generation Computer Systems (in press, DOI 10.1016/j.future.2018.02.042)*.

Baresi, L., Guinea, S., Leva, A., and Quattrocchi, G. (2016). A discrete-time feedback controller for containerized cloud applications. In *Proc. 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*. Seattle, WA, USA.

Boettiger, C. (2015). An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1), 71–79.

Čelik, H. and Sung, K. (2018). Scalable resource allocation for dynamic TDD with traffic and propagation awareness. In *Proc. 2018 IEEE Wireless Communications and Networking Conference*. Barcelona, Spain.

Chen, X., Wardi, Y., and Yalamanchili, S. (2018). Instruction-throughput regulation in computer processors with data-center applications. *Discrete Event Dynamic Systems*, 28(1), 127–158.

Fan, S., Zahedi, S., and Lee, B. (2016). The computational sprinting game. *ACM SIGARCH Computer Architecture News*, 44(2), 561–575.

Ghosh, S., Rajkumar, R., Hansen, J., and Lehoczky, J. (2003). Scalable resource allocation for multi-processor QoS optimization. In *Proc. 23rd International Conference on Distributed Computing Systems*, 174–183. Providence, RI, USA.

Guan, X., Wan, X., Choi, B., Song, S., and Zhu, J. (2017). Application oriented dynamic resource allocation for data centers using docker containers. *IEEE Communications Letters*, 21(3), 504–507.

Kim, C., Nelson, C., et al. (1999). State-space models with regime switching: classical and gibbs-sampling approaches with applications. *MIT Press Books*, 1. Number 0262112388.

Kondguli, S. and Huang, M. (2018). A case for a more effective, power-efficient turbo boosting. *ACM Transactions on Architecture and Code Optimization*, 15(1).

Leva, A. and Maggio, M. (2010). Feedback process scheduling with simple discrete-time control structures. *IET Control Theory & Applications*, 4(11), 2331–2342.

Leva, A., Maggio, M., Papadopoulos, A., and Terraneo, F. (2013). *Control-based Operating System Design*. IET, London, UK.

Leva, A., Seva, S., and Papadopoulos, A. (2018). Progress rate control for computer applications. In *Proc. 2018 European Control Conference*, 3173–3178. Limassol, Cyprus.

Lopez-Novoa, U., Mendiburu, A., and Miguel-Alonso, J. (2015). A survey of performance modeling and simulation techniques for accelerator-based computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(1), 272–281.

Ma, M., Wang, Y., and Li, Y. (2017). Robust output feedback control of positive switched systems with time-varying delays. *International Journal of Advanced Research in Computer Engineering & Technology*, 6(9), 1374–1378.

Maggio, M., Terraneo, F., and Leva, A. (2014). Task scheduling: A control-theoretical viewpoint for a general and flexible solution. *ACM Trans. Embed. Comput. Syst.*, 13(4), 76:1–76:22.

Raghavan, A., Emurian, L., Shao, L., Papaefthymiou, M., Pipe, K., Wenisch, T., and Martin, M. (2013). Utilizing dark silicon to save energy with computational sprinting. *IEEE Micro*, 33(5), 20–28.

Raghavan, A., Luo, Y., Chandawalla, A., Papaefthymiou, M., Pipe, K., Wenisch, T., and Martin, M. (2012). Computational sprinting. In *Proc. 18th IEEE International Symposium on High Performance Computer Architecture*, 1–12. New Orkeans, LA, USA.

Rotem, E., Naveh, A., Ananthakrishnan, A., Weissmann, E., and Rajwan, D. (2012). Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. *IEEE Micro*, 32(2), 20–27.

Seth, S. and Singh, N. (2017). Dynamic threshold-based dynamic resource allocation using multiple vm migration for cloud computing systems. In *Proc. International Conference on Information, Communication and Computing Technology*, 106–116. New Delhi, India.

Suh, C. and Mo, J. (2008). Resource allocation for multicast services in multicarrier wireless communications. *IEEE Transactions on Wireless Communications*, 7(1).

Taylor, M. (2013). A landscape of the new dark silicon design regime. *IEEE Micro*, 8–19.

Terraneo, F., Rinaldi, L., Maggio, M., Papadopoulos, A., and Leva, A. (2014). FLOPSYNC-2: Sub-microsecond and sub-μa clock synchronisation for wireless sensor networks. In *Proc. 35th IEEE Real-Time Systems Symposium*. Rome, Italy.

Wei, G., Vasilakos, A., Zheng, Y., and Xiong, N. (2010). A game-theoretic method of fair resource allocation for cloud computing services. *The journal of supercomputing*, 54(2), 252–269.

Xiao, Z., Song, W., and Chen, Q. (2013). Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE transactions on parallel and distributed systems*, 24(6), 1107–1117.

Yousafzai, A., Gani, A., Noor, R., Sookhak, M., Talebian, H., Shiraz, M., and Khan, M. (2017). Cloud resource allocation schemes: review, taxonomy, and opportunities. *Knowledge and Information Systems*, 50(2), 347–381.

Zahedi, S., Fan, S., Faw, M., Cole, E., and Lee, B. (2017). Computational sprinting: Architecture, dynamics, and strategies. *ACM Transactions on Computer Systems*, 34(4), 12.

Zhang, Q., Zhu, Q., and Boutaba, R. (2011). Dynamic resource allocation for spot markets in cloud computing environments. In *Proc. 4th IEEE International Conference on Utility and Cloud Computing*, 178–185. Melbourne, Australia.

Zheng, W. and Wang, X. (2015). Data center sprinting: Enabling computational sprinting at the data center level. In *Proc. 35th IEEE International Conference on Distributed Computing Systems*, 175–184. Columbus, OH, USA.

Zhu, F. and Antsaklis, P. (2015). Optimal control of hybrid switched systems: A brief survey. *Discrete Event Dynamic Systems*, 25(3), 345–364.