A feedback control approach to dynamic speed scaling in computing systems

Diego Goldsztajn, Andres Ferragut and Fernando Paganini
Universidad ORT Uruguay

***Abstract*—Speed scaling concerns the dynamic adaptation of the active service capacity of a computing system to the processing demands. This problem has received recent attention, motivated by balancing performance with energy consumption; various proposals have been suggested where the processor speed is a function of the current job population, combined with an appropiate scheduling discipline. In this paper we cast the problem in the setting of feedback control, using a fluid model of the queueing system; in this framework the problem is of designing a controller to track the exogenous demand, and the prior work can be seen as restricting the controller to a static function. By allowing for a dynamic controller, in particular a proportional-integral law, we show how the relevant performance tradeoff can be improved. We further indicate a discrete server implementation of this control law, based on a mix of dedicated servers and pooled helpers; its performance is evaluated analytically and by simulation.**

I. Introduction

The possibility of adapting in real-time the processing speed of a computing system has attracted substantial interest in recent years. Original studies of this type [15] were motivated by speed adaptation of hardware, e.g. the microprocessor clock, to trade off processing efficiency with energy consumption. A similar kind of tradeoff appears in the larger scale situation of a data center whose active service capacity is dynamically "right-sized" to match the current load [10]. Yet another incarnation of variable capacity is provided by current-day cloud computing environments such as Amazon EC2 [2]; here virtual machine *instances* with supplementary computing power can be purchased on the fly, thereby adapting the contracted capacity to the real-time demand conditions; [16] provides some analysis of economic tradeoffs in this scenario.

In most of the literature, speed scaling is specified as a function mapping the number of jobs currently present in the system to the active processing capacity, and is combined with a scheduling policy allocating this capacity between the jobs. The analysis of such policies is usually based on a cost that trades off energy and job response time, and comes in two main flavors: a worst-case analysis over a finite batch of jobs [1], [5], [11], or a queueing analysis of a speed scaling system under stochastic demands [3], [7], [14]. This background is briefly reviewed in Section II.

A central requirement, pointed out in [14], is the *robustness* of performance with respect to the system load parameters. Indeed, since demand variations are the main motivation for speed scaling, a solution that requires exact knowledge of the load would not be interesting in practice. Imposing robustness narrows down the options for a good scaling policy, and calls into question some of the favored solutions, as we further discuss in Section II.

In this paper we argue that the performance and robustness requirements are naturally cast in the context of *feedback control*. From this standpoint, speed-scaling is a control system in which the actuated variable is service capacity, and the measurement is job population; the control objective is to track the exogenous load. In Section III we present this perspective using fluid queueing models. In this language, existing proposals amount to using a *static*, possibly nonlinear controller, an observation that helps understand the observed limitations. We suggest that a *dynamic* controller could fare better; in particular we introduce a *proportional-integral* (PI) control, which decouples the latency objective with the robust tracking requirement. We evaluate analytically its performance using fluid models driven by noise in the load process.

In Section IV we take a step towards a more concrete implementation of PI speed scaling in a data center environment, by controlling discrete server units, dynamically summoned to emulate the desired dynamics in the fluid limit. There are baseline units devoted to each job, plus helper units that serve the pool, initially in a processor sharing fashion. We show through simulations that, for systems of moderate scale, performance is close to the fluid predictions.

Section V covers work in progress on further implementation alternatives for the discrete solution. In particular we consider the scheduling component, evaluating the performance of shortest-remaining processing time (SRPT) scheduling for helper units. While analytical work on these ideas is open, simulations look promising. Conclusions and future directions are presented in Section VI.

II. Background and discussion

A seminal reference on the optimization of computing systems with variable speed is [15], which ushered a stream of literature in computer science (e.g., [1], [4], [5], [11]) on the fundamental tradeoff between job latency and energy consumption, for algorithms that control scheduling and speed. The usual formulation is, briefly, as follows: consider a finite set of jobs, and evaluate (i) the total job completion time; and (ii) the total energy expended, assuming that power is a given function $P(s)$ of the service speed s, typically $P(s) = s^\alpha$, $\alpha > 1$. A linear combination of (i) and (ii) is often used as a tradeoff cost. Results take the form of bounding the worst-case *competitive ratio* between the cost achieved by a certain online policy, and the actual minimum cost. The favored policies are the SRPT scheduling, combined with *inverse scaling*, namely setting the speed $s(n) = P^{-1}(n)$ where n is the number of jobs[1]; for instance, [4] establishes this combination is 2-competitive.

[1]Alternatively, $s(n) = P^{-1}(n+1)$ may be chosen.

A parallel approach to this problem is to consider a stochastic queueing system with scalable service capacity. Assuming arriving jobs follow a Poisson process of intensity λ, and service durations are exponentially distributed with parameter μ (i.e. an M/M queue), the model is a continuous time Markov process over the state $n \in \mathbb{N}$ with the following transition rates:

$$q_{n,n+1} = \lambda, \quad q_{n,n-1} = \mu s(n),\ n > 0. \tag{1}$$

The problem of selecting a scaling law $s(n)$ that minimizes an expected cost of the steady-state distribution of (1) was analyzed in [8] through dynamic programming. While the optimal $s^*(n)$ does not have a closed form, bounds can be obtained, as explained below. References [3], [14] apply this solution to the speed-scaling problem, choosing a cost of the form

$$J = E[N] + \frac{1}{\beta} E[P(s(N))]; \tag{2}$$

here N is the random occupation for the queue in steady state, $P(s(N))$ the random service power and β a tradeoff parameter. Through Little's law it is equivalent to trading off mean latency and energy per job. Results apply also to an M/G queue (general job size distribution) under the processor sharing (PS) discipline, exploiting its insensitivity [12].

In [3], [14], numerical comparisons are made between the optimal scaling and two alternatives: the "gated" policy where $s(n) = s_0 \mathbb{1}_{n>0}$, with an adequate choice of s_0, and the *linear scaling* $s(n) = n$, both for PS and SRPT scheduling. One of their main observations is that while there is a slight performance advantage of the optimal law, it suffers from a fundamental *robustness* limitation: the optimal $s^*(n)$ depends critically on the parameter $\rho := \lambda/\mu$ (the system load); departures of the real load from the "design ρ" deteriorate performance, which also happens with the gated policy. In contrast, linear scaling requires no knowledge of ρ and its performance is robust. Moreover, it is far simpler: indeed, in the PS case it amounts simply to an $M/G/\infty$ queue where each job receives its own server. These observations call into question the justification for the more sophisticated versions of speed scaling. We now discuss the issue some more.

Optimal, linear, and inverse scaling

If we impose the robustness requirement that the scaling $s(n)$ must not depend explicitly on the load, then the optimal policy $s^*(n)$ is ruled out, but perhaps there is another possibility that is close to optimal? Inverse scaling $s(n) = P^{-1}(n)$ immediately comes to mind, since it was shown to be 2-competitive in the worst-case model, and does not require knowledge of the load. Furthermore, the bounds given in [14] may suggest it is close to $s^*(n)$. Focusing for simplicity on the case $P(s) = s^2$ and $\beta = 1$, we quote the bounds

$$\sqrt{n - 2\rho} + \rho \leq s^*(n) \leq \sqrt{n} + \rho + \min\left(\frac{\rho}{2n}, C\rho^{\frac{1}{3}}\right),$$

where C is a constant. Another lower bound applies for $n \leq 2\rho$. These bounds imply that indeed $s^*(n) \approx \sqrt{n} = P^{-1}(n)$ as $n \to \infty$.

However, this asymptotic approximation is not relevant since population does not grow without bounds. In fact, since in steady state, the average service rate must balance the load ($E[s^*(N)] = \rho$), typical operation will be with $s^*(n)$ around ρ. In Figure 1 we plot the bounds in [14] for $\rho = 10$, and see that in this region of interest, the inverse law $s(n) = \sqrt{n}$ is far from optimal, whereas linear scaling is in the correct range.

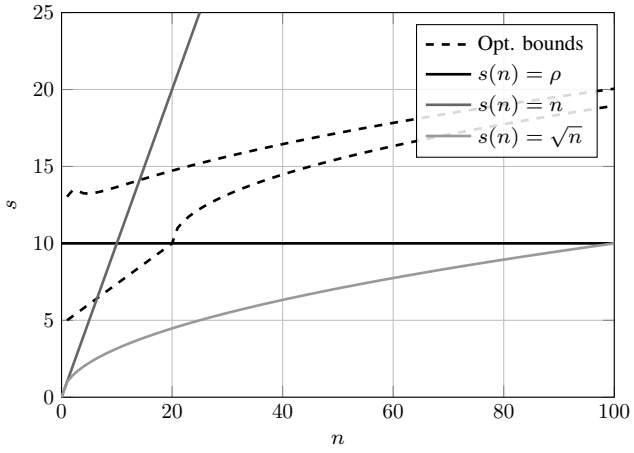

Fig. 1. Bounds on optimal policy, comparison with inverse scaling and linear scaling. $\lambda = 10$, $\mu = 1$

For an analytical comparison of linear and inverse scaling, note that *any* stabilizing policy must satisfy

$$E[s(N)] = \rho. \tag{3}$$

Applying Jensen's inequality to the convex function $P(s) = s^2$ yields $E[P(s(N))] \geq P(\rho) = \rho^2$. Therefore:

(a) The inverse scaling $s(n) = \sqrt{n}$ yields $P = N$, hence $E(N) = E(P) \geq \rho^2$. The combined cost satisfies $J = E[N] + E[P] \geq 2\rho^2$. For instance for $\rho = 10$, we have $J \geq 200$.

(b) Linear speed scaling $s(n) = n$ under PS corresponds to an $M/G/\infty$ queue with a Poisson(ρ) steady-state distribution. Therefore $E[P(S)] = E[N^2] = \rho^2 + \rho$, while $E[N] = \rho$. The overall cost is $J = \rho^2 + 2\rho$, equal to 120 for $\rho = 10$.

So while inverse scaling may be a factor of 2 away from optimality, it is a worse choice than linear scaling in this situation. The reason is that, in order to scale up capacity as required to match the load, the queue is forced to operate around $n \sim \rho^2 = 100$. This degrades performance in terms of latency, without significant energy savings since power has the lower bound ρ^2. The conflict between tracking and performance stems from having forced capacity to be a static function of queue population; the rest of this paper explores the possibility of relaxing this tight coupling.

Remark 1. *The fact that $E[P(s)] \geq \rho^2$ for any policy, means that the power component will dominate the cost J when load is large. This could be compensated by choosing a different tradeoff parameter β, but it seems artificial to tune this parameter to the load. The linear-combination cost that has been favored in the recent literature is in this sense questionable; a more reasonable formulation is perhaps to minimize power subject to a maximum queue occupation $E[N]$, analogous to [15] for the worst-case setting.*

III. Dynamic scaling as feedback control

We introduce here a fluid model of a queue under speed scaling, replacing the Markov chain (1) by the differential equation

$$\dot{n} = [\lambda - \mu s(n) + v]_n^+; \tag{4}$$

This treats $n(t)$ as a continuous quantity, the level of a "tank" with incoming flow λ and "drained" at rate $r = \mu s$. The noise term $v(t)$ will be discussed later on. $[\,\cdot\,]_n^+$ denotes the projection required to keep the dynamics in the positive real line; namely, $[u]_n^+ = 0$ whenever $n = 0$ and $u < 0$, otherwise $[u]_n^+ = u$. A word on units: μ was introduced as a parameter of the job-size distribution; an alternate interpretation (which we will favor) is to say that jobs have mean size 1, and μ in jobs/sec is the baseline capacity of our servers, which in turn can be scaled by the dimensionless quantity $s(n)$.

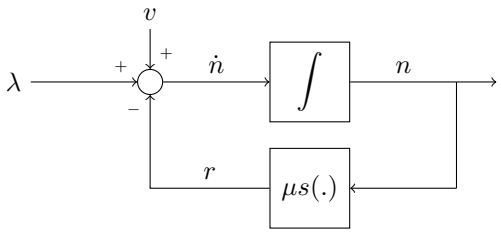

Fig. 2. Speed scaling as static feedback control.

Figure 2 expresses the fluid model in terms of a negative feedback loop. This casts the speed scaling design problem in classical control terms: the "plant" is a (saturated) integrator, whose output n is used as measurement for a feedback controller, with actuation on the service speed r. The loop is set up to track an exogenous variable λ, unknown a priori; the speed-scaling law is the controller. Good performance involves keeping the regulated output n small, keeping in check the "control effort" involved in r. In this language, the speed-scaling strategies considered so far all amount to *static*, possibly nonlinear controllers.

A brief word on the formal relationship between the stochastic and fluid models (see [12]): the solution to (a noiseless version of) (4) can be shown to be the limit of the family of stochastic processes $\frac{1}{k}n^k(t)$ as $k \to \infty$, where n^k is the Markov chain state under arrival rate $k\lambda$, and suitably scaled initial condition. The noise term can be justified by finding the *diffusion* limit of $\sqrt{k}\left(\frac{n^k(t)}{k} - n(t)\right)$; around equilibrium it behaves as a system excited by Brownian motion of variance $2\lambda t$, due to independent contributions of the Poisson arrivals and the departures due to the random job size. In classical terms, the noise $v(t)$ in (4) is white with power spectral density 2λ.

A. Analysis of the static controller

Despite its simplicity, the fluid model (4) contains useful information, starting with the identification of the equilibrium point n^*, which satisfies $\mu s(n^*) = \lambda$; this is the fluid counterpart of condition (3).

The noise model can be used to estimate the variances of relevant quantities around equilibrium. For this we turn to the local linearized model in $\delta n = n - n^*$:

$$\delta\dot{n} = -\mu s'(n^*)\delta n + \sqrt{2\lambda}\; w, \tag{5}$$

Here $w(t)$ is unit white noise. A standard calculation with this first order linear filter yields the variance of the population and speed scaling ($\delta s = s'(n^*)\delta n$) around equilibrium, namely

$$E[\delta n^2] = \frac{\lambda}{\mu s'(n^*)};$$
$$E[\delta s^2] = s'(n^*)^2 E[\delta n^2] = \frac{\lambda}{\mu} s'(n^*).$$

From these expressions one can determine a cost function (analogous to (2)) that trades off mean queue occupation, represented here by n^*, with power, represented by the second moment of s. The cost would be (for $\beta = 1$):

$$J = n^* + (s^*)^2 + E[\delta s^2]. \tag{6}$$

Example 1. *Take the linear scaling $s(n) = n$. From $\mu s(n^*) = \lambda$ we obtain $n^* = \lambda/\mu = \rho$, $s'(n^*) = 1$. Therefore $E[\delta n^2] = E[\delta s^2] = \rho$. The cost in (6) is $J = \rho^2 + 2\rho$, consistent with the result obtained with the stochastic model.*

Example 2. *Take the inverse scaling $s(n) = \sqrt{n}$. From $\mu s(n^*) = \lambda$ we obtain $n^* = \rho^2$, $s^* = \rho$, and*

$$s'(n^*) = \frac{1}{2\sqrt{n^*}} = \frac{1}{2\rho} \Longrightarrow \begin{cases} E[\delta n^2] & = \rho^2/2, \\ E[\delta s^2] & = 1/2. \end{cases}$$

The cost in (6) gives $J = 2\rho^2 + \frac{1}{2}$. In this case we now have an estimate rather than just a lower bound as given before. This estimate was also validated empirically.

B. PI control

Looking at the loop of Figure 2: what a control engineer would consider under these circumstances is, rather than a complicated search over the space of nonlinear static controllers, using a *dynamic* controller.

In particular, since one of our objectives is a small equilibrium queue n^*, the standard prescription is to add another integrator in the controller: in this way the service rate $r = \mu s$ is free to find its correct level to match the input λ, without requiring the queue population to grow. From a dynamic perspective, since having two back-to-back integrators would lead to an oscillatory instability, one must also include a proportional term in the controller. Namely, use proportional-integral (PI) control, a widespread methodology. Figure 3 expresses this proposal in block diagram form.

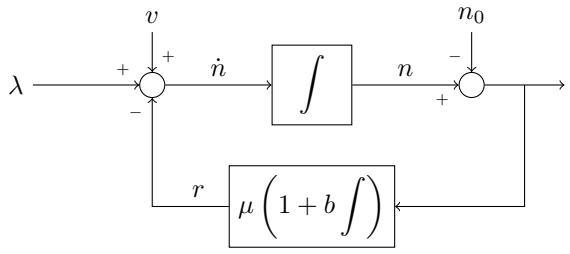

Fig. 3. Proportional-integral control for speed scaling.

In the figure we have included a set-point $n_0 > 0$ in the queue level; the PI controller is based on the error $n - n_0$.

This is required because otherwise, the integrator would have an input of positive sign, implying an increasing output, unable to regulate down if necessary. The latency performance will be determined by the choice of n_0, as further discussed below.

It is not necessary for the *proportional* term to involve n_0. We will actually modify this from the diagram, defining the control loop by the following set of differential equations:

$$\dot{n} = [\lambda - \underbrace{\mu(n+m)}_{r} + v]_n^+; \tag{7a}$$

$$\dot{m} = b[n - n_0]_m^+. \tag{7b}$$

The interpretation is as follows: the speed scaling s has a linear term in n as before, but additional capacity can be summoned through the integrated variable m, as soon as n exceeds the target value of n_0. In this way the service capacity adjusts with no penalty on latency. In the next section we discuss a possible discrete implementation of such a system. Here we will characterize its properties from a fluid perspective.

Imposing the equilibrium condition on (7b) implies $n^* = n_0$. We will assume this target value is low with respect to the load $\rho = \frac{\lambda}{\mu}$, and therefore a positive $m^* = \rho - n_0$ is required to obtain an equilibrium in (7a).

As before, we compute the relevant variances through the linearized model. In state-space form:

$$\begin{bmatrix} \dot{\delta n} \\ \dot{\delta m} \end{bmatrix} = \underbrace{\begin{bmatrix} -\mu & -\mu \\ b & 0 \end{bmatrix}}_{A} \begin{bmatrix} \delta n \\ \delta m \end{bmatrix} + \underbrace{\begin{bmatrix} \sqrt{2\lambda} \\ 0 \end{bmatrix}}_{B} w. \tag{8}$$

The stationary covariance matrix of the state is found (see e.g. [6]) solving the Lyapunov equation $AQ + QA^T + BB^T = 0$, which yields:

$$Q = \begin{bmatrix} \lambda/\mu & 0 \\ 0 & b\lambda/\mu^2 \end{bmatrix}.$$

In particular, the variance of the speed scaling $\delta s = \delta n + \delta m$ is given by

$$E[\delta s^2] = \mathbf{1}^T Q \mathbf{1} = \frac{\lambda}{\mu}\left(1 + \frac{b}{\mu}\right) = \rho\left(1 + \frac{b}{\mu}\right). \tag{9}$$

This is slightly larger than in the linear scaling case (ρ), although we may choose a small $b > 0$ to keep it in check. In compensation, we have a smaller queue and thus a smaller latency: n_0 can be much smaller than the load ρ. The combined cost in (6) in this case is

$$J = n_0 + \rho^2 + \rho\left(1 + \frac{b}{\mu}\right).$$

Again, we note as in Remark 1 that for high-loads the cost will be dominated by ρ^2, which is the *inevitable* portion of the cost, since $s^* = \rho$ is a hard constraint for any stabilizing controller. What we argue here is that, given this necessary power expenditure, the PI controller can achieve a much better performance in job latencies, represented by n_0.

IV. Discrete server implementation

We now discuss an implementation of the proportional-integral control law in terms of discrete servers or computing instances, which is more amenable to implementation in scalable computing environments.

Returning to a stochastic setting, assume as before that jobs arrive as a Poisson process of intensity λ, and job sizes are exponential with mean 1. Upon arrival, each job is allocated a *dedicated* server of rate μ, and we denote by $N(t)$ the random number of jobs (and thus dedicated servers) present in the system at any given time. However, there is a second class of *pooled* servers (for simplicity, also of rate μ) that can be summoned in order to help jobs already in the system. We call these *helpers*, and denote their number at any given time by $M(t)$. In this section we assume that the helper capacity is shared (PS discipline) between all current jobs.

Fig. 4 illustrates the queueing system, with the population N of dedicated servers of rate μ spawned on job arrivals, and the helper capacity $\mu M(t)$ shared among all current jobs.

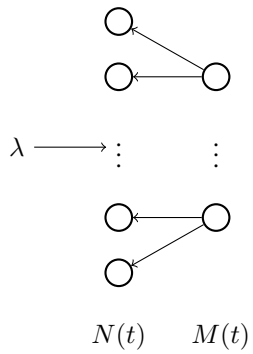

Fig. 4. System with dedicated and helper servers.

Define the following policy for helper creation: each job present in the system summons a helper at rate b while working. Also, a central process decides to recall (destroy) helpers at global rate bn_0 to prevent the number of helpers from growing out of bounds. Assume moreover that helper creation and destruction occur after independent exponential times. Under these assumptions, the process $(N(t), M(t))$ is a continuous time Markov chain with the transition rates depicted in Fig. 5:

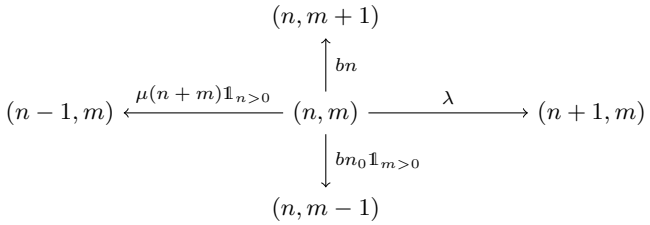

Fig. 5. Continuous time Markov chain implementation of the PI controller.

Note that in these dynamics, the helpers are created by present jobs in order to speed up the system, but the helper removal rate ensures that population stabilizes at a target value n_0 with the appropriate amount of helpers. Fig. 6 contains a simulated trajectory of the Markov chain, for the parameter

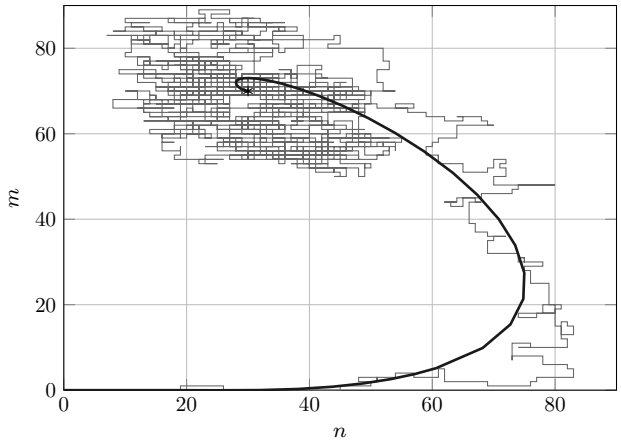

Fig. 6. Markov dynamics and fluid approximation.

values $\lambda = 100$, $\mu = 1$, $b = 0.5$ and $n_0 = 30$. After a transient, we see the dynamics settle around the point $n = 30$, $m = 70$.

We would like to understand this process analytically, and here again we resort to fluid limits. As before, this limit is obtained by replacing transition rates with drift terms in a differential equation, with appropriate projections, as follows:

$$\dot{n} = [\lambda - \mu(n+m)]_n^+ , \tag{10a}$$

$$\dot{m} = b\,[n - n_0]_m^+ . \tag{10b}$$

These fluid dynamics coincide with the PI controller proposed in (7) (for the moment without considering the noise term). In fact, a numerical simulation of the ODE (10) provides the second trajectory depicted in Figure 6, which converges to the equilibrium $n^* = n_0$, $m^* = \rho - n_0$ as expected.

To analyze the variability around equilibrium, once again we will use the diffusion approximation and incorporate noise terms to the differential equation, linearized around equilibrium. There is a difference with respect to the previous section: since helpers are summoned and destroyed at random times, there is an additional noise term injected in the m dynamics, with power spectral density $2bn_0$. The local dynamics with noise is the following variation of (8):

$$\begin{bmatrix} \dot{\delta n} \\ \dot{\delta m} \end{bmatrix} = \underbrace{\begin{bmatrix} -\mu & -\mu \\ b & 0 \end{bmatrix}}_{A} \begin{bmatrix} \delta n \\ \delta m \end{bmatrix} + \underbrace{\begin{bmatrix} \sqrt{2\lambda} & 0 \\ 0 & \sqrt{2bn_0} \end{bmatrix}}_{B} \begin{bmatrix} w^1 \\ w^2 \end{bmatrix},$$

with w^1, w^2 independent unit white noise signals.

Again we compute the steady-state covariance matrix by solving the Lyapunov equation $AQ + QA^T + BB^T = 0$, yielding:

$$Q = \begin{bmatrix} \rho + n_0 & -n_0 \\ -n_0 & \frac{b}{\mu}(\rho + n_0) + n_0 \end{bmatrix}$$

Note that in this case n and m are correlated, something that did not happen for the situation of (7). From the covariance matrix we again find the variance of s:

$$E[\delta s^2] = \mathbf{1}^T Q \mathbf{1} = \rho\left(1 + \frac{b}{\mu}\right) + \frac{bn_0}{\mu}.$$

This is the same as in (9) except for the additional bn_0/μ term. The power cost is higher in this case, however the difference can be made small by an appropriate choice of b. The tradeoff cost becomes:

$$J = n_0 + \rho^2 + \rho\left(1 + \frac{b}{\mu}\right) + \frac{bn_0}{\mu}.$$

V. Scheduling for the discrete implementation

In Section IV we discussed how to implement the PI speed scaling by spawning individual computing instances. We now take a closer look at the allocation of this pooled capacity among present jobs. In particular we are concerned about practicality of implementation, and the possibility of further improving performance. In regard to implementation, the PS policy previously discussed can be impractical because each helper instance must divide its rate between all present jobs. Such fine grained scheduling may be infeasible in large scale problems, so we now analyze a simpler alternative showing the same behavior.

Consider the following policy: assume each time a helper instance spawns, it is assigned to a random job in the system (with possibly multiple helpers attending a single job). Every time a job finishes, the helpers working on it can be rescheduled to another random job. Helpers are destroyed as before at a rate bn_0. This *random assignment* policy is very easy to implement, and since the helper assignment is work-conserving (no helper servers are idle), the total available rate is used, and the total population of jobs and helpers follows again the model in Fig. 5. By performing this random assignment, we avoid the need to explicitly pool the helper capacity, and the randomness ensures that this capacity is fairly distributed among jobs currently in service.

In Fig. 7 we plot the time evolution of the system under the PS discipline and under the random assigment policy, with the fluid model solution (10) for reference. We can see that performance is similar in both cases.

A different approach is to try to redirect the extra capacity towards finishing as much jobs as possible thus minimizing delay. In the context of fixed capacity systems, it is well known [9], [13] that the shortest remaining processing time (SRPT) discipline is optimal in terms of minimizing delay. In the context of dynamic scaling, no such result is available, and in fact simulations show that the improvement is not much over PS when using linear or inverse scaling [14].

Nevertheless, it is worth examining whether allocating helper capacity using SRPT scheduling can improve system performance. Note that in this case, since the scheduling is determined using full knowledge of job remaining time, the stochastic system no longer behaves as a Markov chain, and the fluid limit is also no longer a valid approximation. We thus resort to simulation: results under the same parameters are shown in Fig. 8. The fact that helper capacity is created to stabilize the population at $n = n_0$ (and thus, by Little's law delay is stabilized), makes the contribution of SRPT less important in terms of delay. However, as we can see in Fig. 8, transients are damped, and variability in service rate is reduced, thus obtaining a more predictable usage profile, a desirable feature in provisioning of data centers or cloud instances.

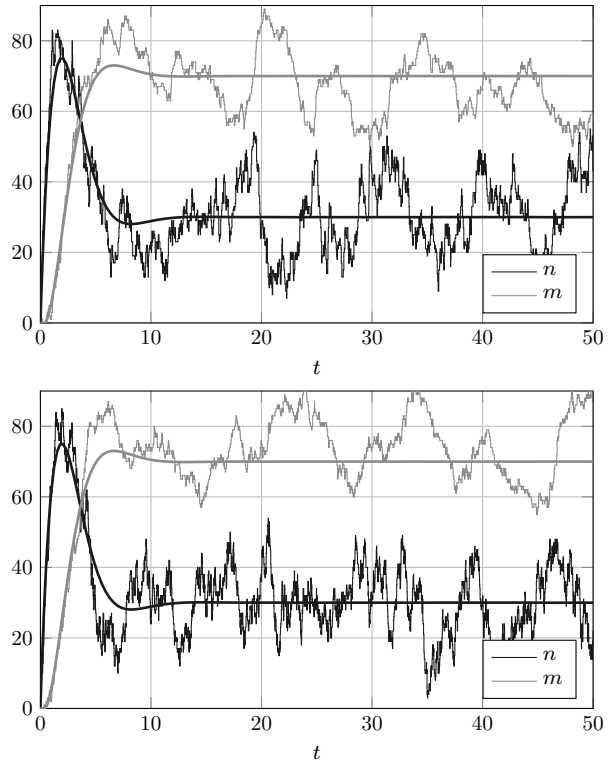

Fig. 7. PS (above) and random assignment (below) time evolution with dynamic scaling. $\lambda = 100$, $\mu = 1$, $b = 0.5$ and $n_0 = 30$.

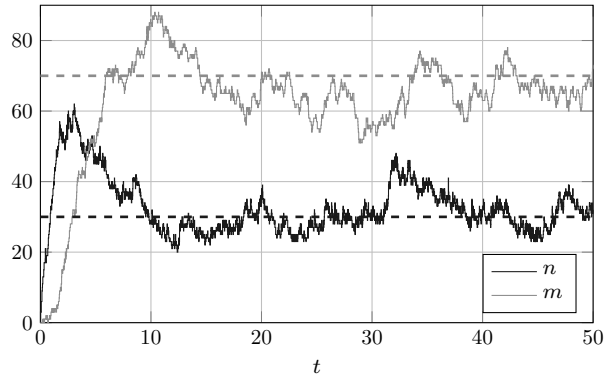

Fig. 8. SRPT scheduling for helper capacity with dynamic scaling, time evolution for $\lambda = 100$, $\mu = 1$, $b = 0.5$ and $n_0 = 30$.

On the flip side, implementing an SRPT scheduling mechanism involves not only pooling the whole helper capacity, but also obtaining precise information of remaining processing time across all jobs, in order to redirect capacity. Finding a practical approximation of the SRPT policy in the context of large scale systems will be pursued as future work.

VI. Conclusions and future work

We have analyzed the problem of speed scaling in computing systems from the point of view of feedback control. From this perspective, prior work on this problem is restricted to static, memoryless controllers based on the current queue occupation. Allowing for a dynamic controller with modest complexity (PI) has advantages in terms of achieving robust tracking of an exogenous load, while maintaining job latencies in check. A discrete implementation of such controller was proposed, in which the integral term is implemented through a pool of helper servers. It is shown how performance is adequately predicted by the fluid models considered. We also present preliminary work on the scheduling component for these pooled servers, with particular attention to SRPT scheduling. In future work we intend to delve more deeply into these implementation alternatives.

Acknowledgment

The authors would like to thank R. Srikant for his input in the early stages of this work. This research was partially supported by AFOSR US under grant FA_9550_15_1_0183.

References

[1] S. Albers and H. Fujiwara, "Energy-efficient algorithms for flow time minimization," *ACM Transactions on Algorithms (TALG)*, vol. 3, no. 4, p. 49, 2007.

[2] Amazon Web Services, "Amazon Elastic Compute Cloud (EC2)." [Online]. Available: http://aws.amazon.com/ec2/

[3] L. L. Andrew, M. Lin, and A. Wierman, "Optimality, fairness, and robustness in speed scaling designs," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 1, pp. 37–48, 2010.

[4] L. L. Andrew, A. Wierman, and A. Tang, "Optimal speed scaling under arbitrary power functions," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 2, pp. 39–41, 2009.

[5] N. Bansal, H.-L. Chan, and K. Pruhs, "Speed scaling with an arbitrary power function," in *Proceedings of the twentieth annual ACM-SIAM symposium on discrete algorithms*. Society for Industrial and Applied Mathematics, 2009, pp. 693–701.

[6] J. Burl, *Linear Optimal Control: $\mathcal{H}_2$ and $\mathcal{H}_\infty$ methods*. Menlo Park, CA: Addison Wesley, 1999.

[7] L. Chen and N. Li, "On the interaction between load balancing and speed scaling," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2567–2578, 2015.

[8] J. M. George and J. M. Harrison, "Dynamic control of a queue with adjustable service rate," *Operations Research*, vol. 49, no. 5, pp. 720–731, 2001.

[9] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

[10] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 5, pp. 1378–1391, 2013.

[11] K. Pruhs, P. Uthaisombut, and G. Woeginger, "Getting the best response for your erg," *ACM Transactions on Algorithms (TALG)*, vol. 4, no. 3, p. 38, 2008.

[12] P. Robert, *Stochastic networks and queues*. Springer Science & Business Media, 2013.

[13] L. Schrage, "Letter to the editor—a proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, vol. 16, no. 3, pp. 687–690, 1968.

[14] A. Wierman, L. L. Andrew, and A. Tang, "Power-aware speed scaling in processor sharing systems: Optimality and robustness," *Performance Evaluation*, vol. 69, no. 12, pp. 601–622, 2012.

[15] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*. IEEE, 1995, pp. 374–382.

[16] L. Zheng, C. Joe-Wong, C. G. Brinton, C. W. Tan, S. Ha, and M. Chiang, "On the Viability of a Cloud Virtual Service Provider," in *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*. ACM, 2016, pp. 235–248.