Queue Response Times with Server Speed Controlled by Measured Utilizations

Murray Woodside(✉)

Carleton University, Ottawa, Canada
cmw@sce.carleton.ca

Abstract. Because CPUs use speed control to conserve energy their response times may be greater at low loads, than if they were operating at full speed, giving a flatter response curve against load and trading off longer response time at light loads for energy savings. When conservative control is applied to average utilizations and the averaging time is too long, a different and somewhat "toxic" response-time curve results instead. A numerical investigation was undertaken of open and closed single-server queues with server speed controlled by feedback of CPU utilization measures, which is a common approach for CPUs. With higher target utilizations and longer averaging times for the measured utilization, an undesirable non-monotonic pattern (rising response time, then falling, finally rising again) emerges. This gives unstable behaviour and could disrupt autoscaling strategies that assume monotonically increasing response times. Recommendations have been found for controller parameters, to avoid non-monotonic response times. It is concluded that speed control based on measured utilizations has limited usefulness if performance is a concern, which is in line with industry recommendations. Better speed governors are needed.

Keywords: Performance management · Performance model · Controlled queue

1 Introduction

Modern computer processors have a controllable clock, which can be set by a strategy implemented in the operating system to speed up or slow down the processor, and save energy at lower speeds. The classical queuing models for processor performance need to be adapted because they assume a known constant service rate. This note investigates the response times for a commonly used feedback control strategy and finds that the response time may follow a desirable flattened pattern, or a highly undesirable non-monotonic zig-zag shape which could reduce the effectiveness of some adaptive mechanisms, depending on the speed control parameters. While standard settings may favour the former, more aggressive power-saving will move the parameters towards the non-monotonic behaviour and may cause unexpected system instability, and disrupt adaptive application management. The purpose of this work is to understand the phenomenon and to characterize the parameters that give the desirable and undesirable patterns.

A. Abate and A. Marin (Eds.): QEST 2021, LNCS 12846, pp. 295–309, 2021.
https://doi.org/10.1007/978-3-030-85172-9_16

Speed control has been studied in many different ways, but not for feedback of performance measures. For a general stochastic workload, optimal stochastic control provides policies for the optimal speed as a function of the queue state, as in the work of Lu et al. [10] to optimize a combination of power and response time. These authors also give references to other research in this direction. While this gives an optimal solution to the problem considered here, feedback of queue state is not commonly used in practice, as described below. The purpose here is not to find the ideal control but to model the effect of policies that are used in practice.

Optimal policies have also been found for problems in which the system is more narrowly specified. For example, in [9] Li et al. consider controlling processing speed, memory latency and memory bandwidth on multiple processors to minimize the makespan of a predefined parallel processing job across the cores of a processor. In [13], Rao et al. consider controlling the overheating effect of completing a given total computation in minimum time, leading to an optimal profile of speed values over time. In [12], Mutapcic et al. determine optimal speed values for heterogeneous co-located processors that share a heat sink. In [7] speed control is used to help achieve task deadlines, and also to deliver video frames on time for video streaming (to compensate for the variation in the processing to be done per frame).

The technology of power management is described by Gough, Steiner and Saunders [6]. Processors offer a set of clock speeds (called P-states by Intel [8]) which can be set by the operating system. Linux has a set of standard speed-control policies called governors [5]. Many governors have been implemented (a list of 117 of them with short descriptions is given in [14]), but three are notable:

- "performance" [3, 6], which runs at maximum speed when the CPU is busy and switches to minimum when it is idle,
- "conservative" [3] which steps the speed up and down based on average utilization,
- "schedutil" [3], an evolved version similar to "conservative" which uses moving-average load measures formed by the scheduler, related to utilization.

Here the "conservative" governor is modeled, since it represents the strongest effort to save power. These controllers are for Linux; similar controllers are used by the Android [1] and Windows [11, 6] operating systems. Processors are also capable of short-term additional speed (called "turbo" operation) which cannot be sustained due to overheating; this work does not consider non-sustainable speeds.

A CPU with the "conservative" governor is modeled below as a single server with open and closed Markovian workloads and control via a speed reduction factor S that can be set in the range $S_{min} < S < 1.0$. The controller measures the server utilization U over an interval and then raises S by a step if U is too high, or lowers it if U is too low. No previous study of queues with this family of speed control policy could be found, which motivates this report.

A Naive Motivating Observation

Suppose a queue has a target value of utilization of U^* and a service rate $S\mu$ with $S_{min} < S < 1.0$. Figure 1 shows the two response-time functions for the extreme values of S, and two paths that might be enforced by speed control. A-AA-C-D is a desirable case which

maintains a constant response time over a wide range of loads. However a controller might instead give a curve like A-B-C-D, if it enforced the principles:

- maintain S_{min} as long as the utilization is below U^*
- raise the speed to prevent the utilization from exceeding U^* if possible

An adaptive system controller that scales up servers depending on R might be trapped in the segment AB and deploy unnecessary extra servers, rather than exploit the improved response available around C.

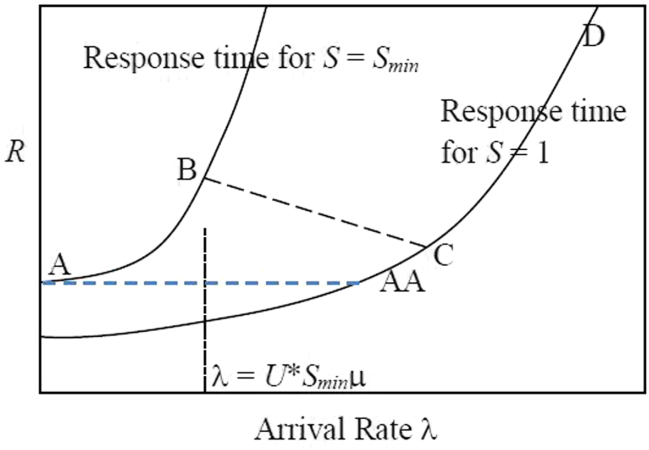


Fig. 1. Naive idea of possible response curves

In real systems the speed is controlled in steps at discrete control instants, and the control is based on past measured utilizations rather than on exact knowledge. This introduces both a control delay and statistical estimation errors. A Markov model with these features is developed below, with numerical solutions. In Sect. 4 the naive view above is formalized as an idealized "perfect knowledge" (PK) approximation to $R(\lambda)$. Both the approximation and the Markov model have the zigzag form seen in Fig. 1.

An Experiment

A simple experiment shows that actual controllers have been adjusted to give desirable flattening, but also confirms the existence of non-monotonic response time functions. A closed concurrent workload with 300 programs was run in parallel on a laptop under the "power saving" power management option which controls the processor speed to a target utilization. The average response times shown in Table 1 initially rise, drop between throughputs of 5000/s and 8700/s, and then rise again. The measured values are accurate to about 2% so the dip is statistically significant, although small.

Table 1. Measured response times on a microsoft windows personal computer

Think time between operations (sec)	Throughput (responses/sec)	Mean response time R (sec)	Confidence interval for R ($\pm$)
0	22561	0.01330	0.00031
0.025	8754	0.00927	0.00024
0.05	5024	0.00971	0.00015
0.075	3547	0.00957	0.00028
0.15	1881	0.00950	0.00027

2 The Model

We consider a single server with one class of customers with arrival rate λ and exponentially distributed service demand of mean μ^{-1}. Its speed factor S gives a mean service rate of $S\mu$, and takes values $S(1)$, … $S(s_{max})$ indexed by an integer "speed index" s, with $S(1) = S_{min}$ and $S(s_{max}) = 1$. A utilization estimate $\widehat{U}$ is formed by averaging over a measurement interval of length Δ sec, and is compared to lower and upper thresholds U^- and U^+, which bracket the target value U^*. The control law adjusts the speed index s according to:

- if $\widehat{U} < U^-$, decrease s by one, stopping at the minimum value 1,
- if $\widehat{U} > U^+$, increase s by one, stopping at the maximum value s_{max}.

To make the solution numerically practical a discrete-time version was adopted with a time-step of length δ. In each time-step there is an arrival with probability $\lambda\delta$ and a departure (if the server is busy) with probability $\mu\delta$. For small values of δ this approximates Poisson arrivals at rate λ and exponential service at rate μ. To make the state space finite the state n was truncated at N and arrivals in state $n = N$ are lost.

The decision interval was taken as a multiple $K\delta$ of the time-step δ. The utilization is measured by counting the number b of the K steps that have a busy server, and is compared to lower and upper target values KU^-, KU^+. For the decision, if s_0 is the previous index, the new value $s_{new} = s_{new}(s_0, b)$ is given by:

Speed-Control Algorithm:

$$\begin{array}{ll} \texttt{if} & b < KU^- : s_{new}(s_0, b) = \max(s_0 - 1, 1) \\ \texttt{elseif} & b > KU^+ : s_{new}(s_0, b) = \min(s_0 + 1, s_{max}) \\ \texttt{else} & : s_{new}(s_0, b) = s_0. \end{array}$$

Because s changes only at decision points, the model can be decomposed into two Markov Chain models, a transient chain one to model the states between decision points, and the other embedded at the decision points, to model the transitions in s

(1) The lower level Discrete-Time Markov Chain (DTMC) [2] models the transient behaviour of the queue state and the accumulated busy time over one interval between decision points. It has:

- s_{max} submodels, one for each value of s,
- each with a state (n, b) at substep k, where $n \in [0, ..., N]$ is the number of customers in the queue and $b \in [0, ..., K]$ is the accumulated busy time of the server,
- an initial state $(n_0, 0)$, where n_0 is the value of n at the decision point that begins the interval.

The lower-level DTMC is solved for each combination of s and n_0 to determine the probability $p(n, b; k, n_0, s)$ for state (n, b) at substep k of the interval. The model equations and solution method are conventional [2]; the important result is the final probability $p(n, b; K, n_0, s)$, and the mean number of customers over the transient, denoted as $\overline{n}(n_0, s)$

(2) The upper level DTMC models the joint state (n, s) of the queue and the speed controller, embedded immediately after the decision points for the control algorithm. It has transition probabilities $A(n_0, s_0; n, s)$ for a transition from state (n_0, s_0) immediately after a decision, to the successor state (n, s) immediately after the next decision K substeps later. The steady-state probabilities of this DTMC are denoted $\pi(n, s)$.

.

The transition probabilities A for the upper level model are found from the solutions of the lower level model. Let $B(s; s_0, b)$ be the set of utilization measures b such that the speed-control algorithm above gives $s_{new}(s_0, b) = s$. Then

$$A(n_0,\ s_0; n, s) = \sum\nolimits_{b \in B(s, s_0, b)} p(n, b; K, n_0, s) \tag{1}$$

The upper-level DTMC with $(N + 1)s_{max}$ states was solved for the steady-state probabilities $\pi(n, s)$ for the number in queue and the speed level, after a decision point. The equations and the solution are conventional [2]. The steady state mean number in the queue is found by combining the results for the transients, conditioned on the initial state, giving the overall mean number in the queue $\overline{\overline{n}}$ and overall mean arrival rate (without the lost arrivals) of $\overline{\overline{\lambda}}$:

$$\overline{\overline{n}} = \sum_{n_0, s_0} \overline{n}(n_0, s_0)\pi(n_0, s_0)$$

$$\overline{\overline{\lambda}} = \sum_{n_0, s_0} \overline{\lambda}(n_0, s_0)\pi(n_0, s_0)$$

The mean response time for the given arrival rate, service rate, speed factor steps and utilization thresholds is then given by:

$$R = \overline{\overline{n}} / \overline{\overline{\lambda}} \tag{2}$$

3 The Solution

To explore the form of the solution, the major parameters were varied across their possible range of values. The service rate μ was normalized to 1/sec, and.

- $U_0 = \lambda/\mu$ (the high-speed utilization) was varied from 0.1 to 0.9;
- the target server utilization U^* was varied from 0.3 to 0.9;
- the steps in speed were set to the values [0.4, 0.6, 0.8, 1.0] and
- the control interval Δ took values [1, 10, 30, 100, 300] sec.

Other parameters were:

- the utilization thresholds $U^- = 0.9$ U* and $U^+ = \min(1.1U^*, 0.95)$.
- the time-step $\delta = 0.2$ s. in the lower-level model
- a limit of 60 queue states, giving a limit of N = 59 customers

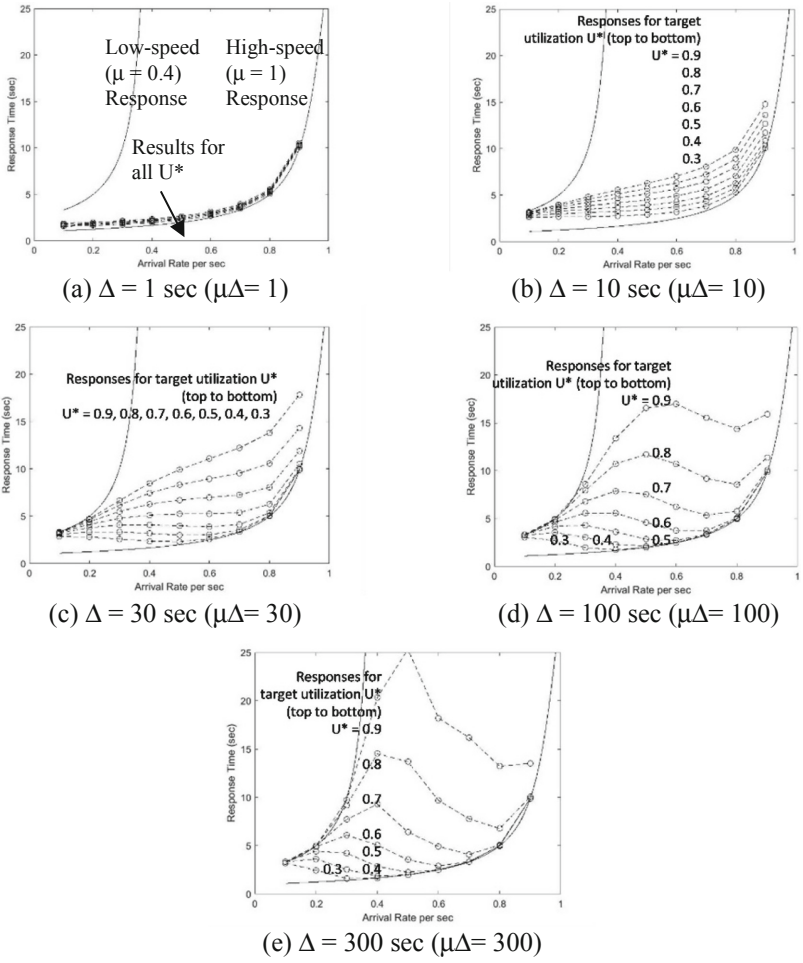
Some results giving a broad picture of the solution are shown in Fig. 2. The computed response times are plotted as circles, and the boundary cases of the maximum-speed and minimum-speed response times are plotted as solid curves for reference.

The smaller is Δ, the closer R stays to the high-speed asymptote. For higher Δ larger U^* gives larger response times, and the response curves are almost flat. For $\Delta = 100$ and 300 the zigzag shape predicted in Fig. 1 emerges and has a very pronounced peak. The "desirable" flat response pattern is obtained for cases with $\Delta \leq 30$ and pronounced "undesirable" peaking is obtained for cases with larger Δ and for U* > 0.5.

The cause of the undesirable peaking can be traced to two factors that are both linked to Δ, the control delay. For small Δ the control delay (which is at least Δ) is small but the estimation accuracy is low, while for large Δ the accuracy is better but the control delay is large. Clearly the effect of a large control delay overwhelms the system. Consider this scenario: a combination of stochastic load fluctuations and estimation errors puts the server into a low-speed state which leads to a long unstable transient increase in the queue until the end of the next estimation/control period.

In uncontrolled queueing systems, heavier loads always lead to longer response times, and many adaptive scaling algorithms for computer systems are based on this. The survey in [4] found that the largest number of reported autoscalers use this kind of simple feedback loop. We can imagine a scaling algorithm based on response time, which increases the capacity of a heavily loaded system that is operating to the right of the peak. This would give increased response times as the load on each CPU is reduced, and therefore it would scale up further until it crosses the peak into the less economical regime to the left, and then be stuck there.

Worse, the value of Δ is difficult to tune. Δ is the ratio of the measurement interval to the mean service time of CPU requests, which varies with the program being executed, so a CPU could be driven randomly between the desirable pattern in parts (a)–(c) and the less desirable pattern in parts (d) and (e).



(a) Δ = 1 sec (μΔ= 1)

(b) Δ = 10 sec (μΔ= 10)

(c) Δ = 30 sec (μΔ= 30)

(d) Δ = 100 sec (μΔ= 100)

(e) Δ = 300 sec (μΔ= 300)

Fig. 2. Exact (numerical) response times

Two practical recommendations emerge from Fig. 2. First, if $U*$ is set to a mid-range value such as 0.5 it tends to give a wide plateau at near-constant response time, for a wide range of Δ. Second, if Δ is set at a moderately small value such as 10 times the mean service time (Δ = 10 in Fig. 5)) the response time is attractively flat, while a long averaging time (300 times the mean service time) gives unstable behaviour and possibly very long responses (at the peak). However the longer averaging time also gives lower control overhead and (as shown below) lower average power.

4 An Idealized "Perfect Knowledge" (PK) Analysis

An idealized analytic model was created by assuming that:

1. the controller has perfect knowledge of λ, μ, and U over the next interval ($\widehat{U} = U$)

2. the thresholds are equal ($U^- = U^+ = U^*$) and
3. the estimation interval Δ and the size of the steps in values of S approach zero,

Define $U_0 = \lambda/\mu$ and $S^* = U_0/U^*$, which is the speed factor which would make $U = U^*$ if it is feasible. Then the PK control law uses S^*, constrained to the range (S_{min}, 1):

$$S = f(U_0) = max\big(S_{min}, min\big(1.0, U_0/U^*\big)\big) \tag{3}$$

We will assume a queueing discipline for which the response time has an analytic solution of the form:

$$R(\lambda, \mu) = C/(S\mu - \lambda) = C/(f(U_0)\mu - \lambda) \tag{4}$$

for some constant C; these disciplines include processor sharing (with $C = 1$), which is an approximation to real time-slicing disciplines, and M/G/1, in either case with Poisson arrivals and general service processes [2]. Using Eq. (4) this can be written as:

$$\text{PK approximation: } R_{PK}(\mu, \lambda) = \min(R_1(\mu, \lambda), \max(R_2(\mu, \lambda), R_3(\mu, \lambda))) \tag{5}$$

where

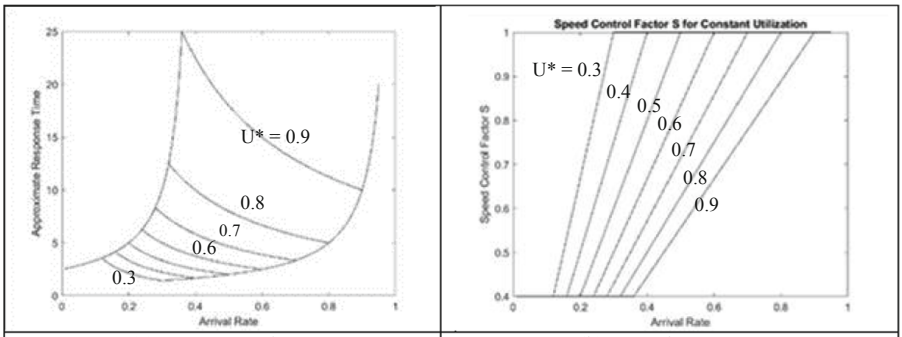
- $R_1(\mu, \lambda)$ is the response time with $S = S_{min}$: $R_1(\mu, \lambda) = C/(S_{min}\mu - \lambda)$
- $R_3(\mu, \lambda)$ is the response time with $S = 1$: $R_3(\mu, \lambda) = C/(\mu - \lambda)$
- $R_2(\mu, \lambda)$is the response time with $S = S^*$: $R_2(\mu, \lambda) = CU^*/(\lambda(1 - U^*)$

Since U_0/U^* is monotonically increasing in λ, R is unique for each λ and there are three regimes in which $R_{PK}(\mu, \lambda)$ equals $R_1(\mu, \lambda)$, $R_2(\mu, \lambda)$, $R_3(\mu, \lambda)$ in turn. They are separated by two thresholds λ_1 and λ_2 in the arrival rate, which define the points where U_0/U^* reaches its limits of S_{min} and 1. If U_0/U^* is always greater than S_{min}, λ_1 is zero, and if it is always less than 1, λ_2 is set to infinity. This is summarized in the following table (Table 2).

Table 2. Definition of the PK-approximation

Regime	Condition on λ	S given by	Response time $R(\lambda, \mu)$ given by
AB: low speed	$\lambda \le \lambda_1 = \mu U^* S_{min}$	$S = S_{min}$	$R_1 = C/(S_{min}\mu - \lambda)$
BC: controlled speed	$\Lambda_1 < \lambda \le \lambda_2 = \mu U^*$	$S = S^* = U_0/U^* = S_{min} + (\lambda - \lambda_1)/(\mu U^*)$	$R_2 = C/(S\mu - \lambda)$
CD: high speed	$\lambda > \lambda_2$	$S = 1$	$R_3 = C/(\mu - \lambda)$

For an M/M/1 queue (for which $C = 1$) with nominal service rate $\mu = 1.0$ and $S_{min} = 0.3$, the response time approximations for varying λ and some different settings of the



(a) Response times (b) Speed Factor

Fig. 3. The PK-Approximation response time and speed control factor for an open single server queue with various utilization targets, with $S_{min} = 0.3$

target utilization U^* are shown in Fig. 3(a). They strongly resemble Fig. 1. Figure 3(b) shows the speed control settings. Since the power used increases with S it is evident that the higher U* is set, the less power is used.

In the limit as $\mu\Delta \to 0$, $\hat{U}$ will be either zero (if the server is idle) or 1 (if it is busy) and the control will switch between S_{min} for $\hat{U} = 0$. and $S = 1$ for $\widehat{U = 1}$; thus whenever there is a customer the server will run at maximum speed and:

$$\text{Zero} - \text{averaging} - \text{interval asymptote} : R(\mu, \lambda) = R_3(\mu, \lambda) \tag{6}$$

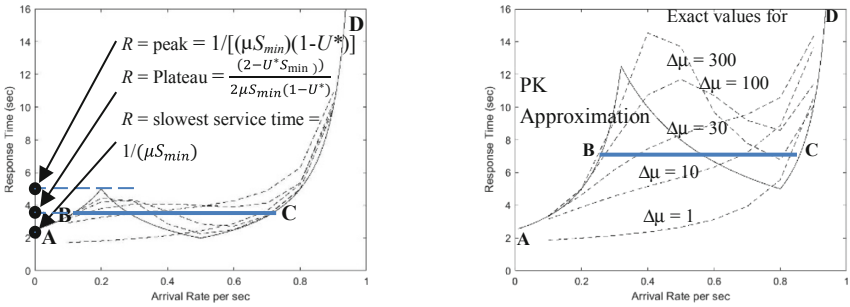
This corresponds to the "performance" governor in Linux [5].

The accuracy of the PK approximation is quite good for some situations and poor in others. Figure 4 compares it to the numerical exact results. For moderate target utilization ($U^* = 0.5$) it is quite accurate for a wide rage of values of the normalized estimation time $\mu\Delta$. For a higher value ($U^* = 0.8$) it is however quite poor. The hump in the response time curve corresponds in positioning and amplitude for intermediate values of $\mu\Delta$, but is more pronounced in the exact results for large U* a large $\mu\Delta$.

In general the exact response time is below the approximation (sometimes much below) for low loads and above it for high loads. This is due to the estimation errors in $\hat{U}$; when the ideal speed is near a boundary (S_{min} or S_{max}) the estimation errors tend to diffuse the speed away from the boundary. The exaggerated hump for large $\mu\Delta$ is probably due to the larger control delay which allows the queue to build up when the system accidentally (due to estimation error) enters a state with low speed and high load. For long averaging times ($\mu\Delta = 100$ and 300) the mean relative absolute error (MRAE) was 9.5%. Over all cases the MRAE was 33%, and it was particularly high for short averaging and large U^*.

A Usable "Plateau" Approximation

For moderate values of U^* up to about 0.5 the flattening in the response time curve can be described by a simpler "plateau approximation" with a constant response in the middle range between the curves for S_{min} and for $S = 1$. Figures 4(a) and 4(b) show it as a bold line BC placed halfway between the response time at zero load ($R = 1/(\mu\ S_{min})$)



(a) $U^* = 0.5$ (b) $U^* = 0.8$

Fig. 4. The plateau approximation ABCD for response time

and the response time at the peak of the PK approximation, which can be shown from Eq. (7) and (8) to be $R = 1/[(\mu\ S_{min})(1 - U^*)]$. The plateau is approximated by their average, given by.

$$\text{Plateau} : R_P(\mu, \lambda) = \left(2 - U^* S_{min}\right)/2\mu S_{min}\left(1 - U^*\right)$$

Combining this with the low and high-speed boundary curves the entire plateau approximation is given by

$$\text{Plateau approximation} : R(\mu, \lambda) = \min(R_1(\mu, \lambda),\ \max\ (R_2(\mu, \lambda), R_P(\mu, \lambda)) \quad (7)$$

(where R_1 and R_2 are given by Eq. (4)), and is shown as the curve ABCD in Figs. 4(a) and 4(b). For the cases studied that have target utilization $U^* \leq 0.5$ and normalized estimation interval $\mu\Delta \leq 100$, the MRAE was 11.8%.

To summarize the various possible cases, approximate values can be computed, with average errors around 10% in these situations:

- for very small $\mu\Delta$ by using the high-speed response time,
- for moderate $\mu\Delta$ and U^* by the Plateau approximation Eq. (7), and
- for large $\mu\Delta$ by the PK approximation, Eq. (5).

For moderate $\mu\Delta$ and large U^* a useful approximation has not been found.

5 Effectiveness of Utilization Control and Power Saving

The controller is driven by utilization values and a natural question is, how close does it stay to the target U^*? Figure 6 shows utilization results for the same queue with U^* = {0.3, 0.4, 0.5, 0.6 0.7, 0.8, 0.9}. Following the bottom dashed curve for $U^* = 0.3$ we can see that it does not dwell at $U = 0.3$ for any substantial interval of arrival rates, unless $\mu\Delta$ is at least 100. Thus, the utilization is not well controlled.

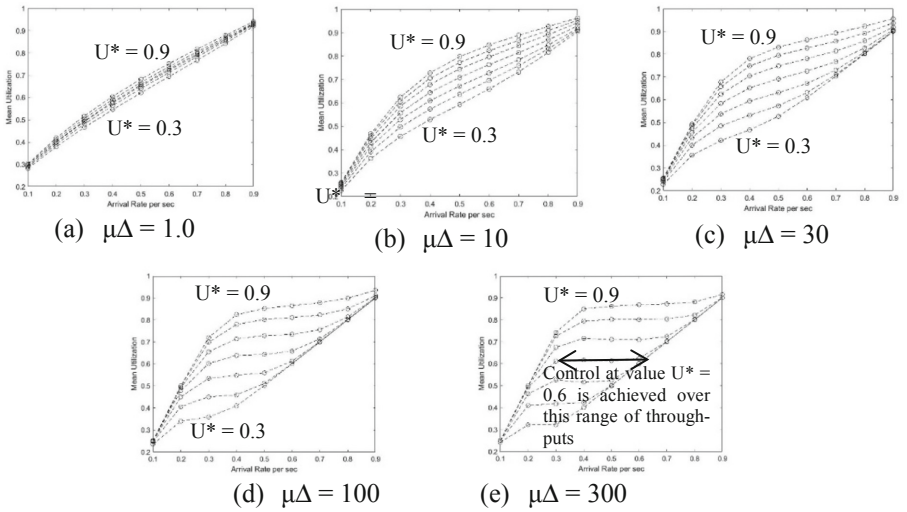


Fig. 5. Utilizations for the same cases as in Fig. 2. Values of U* increase from bottom to top, from 0.3 to 0.9 in steps of 0.1.

Figure 5shows that only for $\mu\Delta = 300$ is there substantial success in achieving the target utilization over a range of throughputs. For $\Delta = 100$ there is some success, and for shorter averaging times very little.

How effective is this form of speed control at reducing power? The SPECPower benchmark measures power consumption as a function of processing speed, as illustrated in Fig. 6. These results will be used, normalized to the maximum speed taken as $S = 1$, to estimate the power consumption based on S. For the processor in Fig. 7, taking a line from the point at zero ops (assumed to correspond to $S = 0$) to the maximum reported speed ($S = 1$) gives as a rough approximation:

$$\text{Power} = 11.6 + 33.1S \tag{8}$$

Using Eq. (8) and the mean value of S found from the model solution gives the power levels shown in Fig. 7. The dashed lines curves are for values of U^* in the range 0.3–0.9 as before, increasing from top to bottom; the solid lines are for $S = 1$. The available power savings are negligible at high loads, about 20% at $U_0 = 0.5$ and about 50% at $U_0 = 0.2$, compared to the solid line. Notice that as long as the power function is linear and increasing, the conclusion that one condition uses less power than another is unaffected by the particular values of the coefficients in Eq. (8).

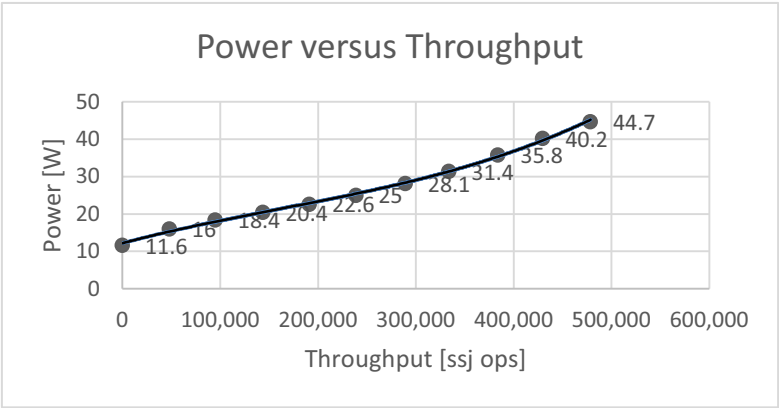
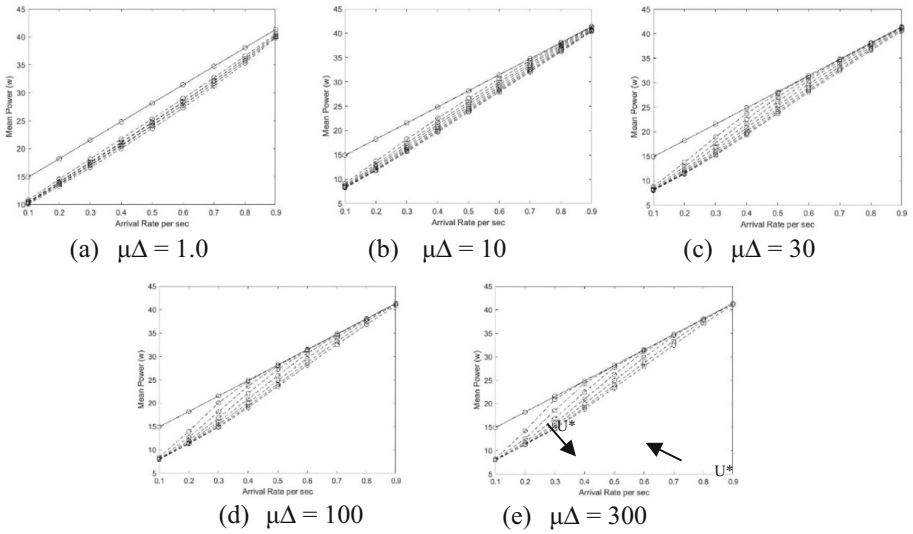


Fig. 6. Power versus throughput for Fujitsu Server PRIMERGY TX1320 M2 [15]



(a) $\mu\Delta = 1.0$ (b) $\mu\Delta = 10$ (c) $\mu\Delta = 30$

(d) $\mu\Delta = 100$ (e) $\mu\Delta = 300$

Fig. 7. Average power in watts, for the cases shown in Fig. 2, using the power approximation in Eq. (8). Values of U^* increase from top to bottom.

6 Control of Finite-Population Queues

A finite-population model may be preferred since a processor usually has a finite number of processes or threads that may request to be scheduled. A corresponding PK approximation can be constructed using the well-known solution of the M/M/1//N queue, giving the solutions displayed in Fig. 8. To compare the approximation for the closed and open cases the same example was analyzed with a finite population of 60 customers, and $0.3 < S < 1.0$. The arrival rate r for each customer not at the server ranged from 0 to

0.03 which gave a similar range of throughput values from 0 to 1, as in the open cases examined above.

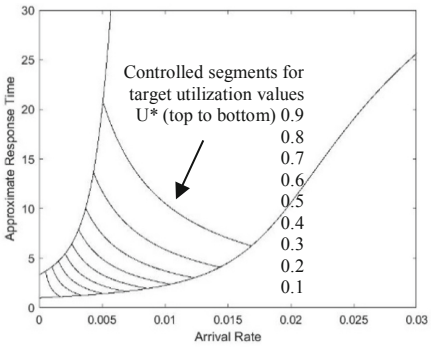
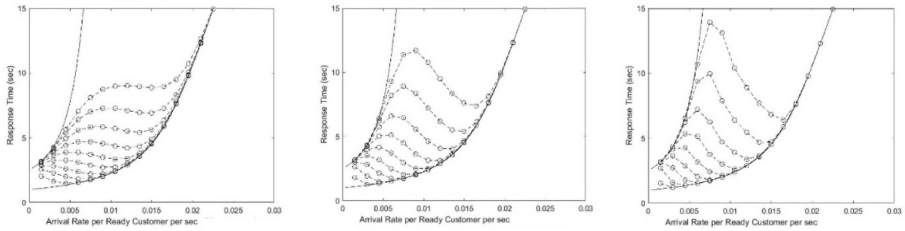


Fig. 8. Approximate "perfect knowledge" response time curves for a closed queue with 60 customers

As the arrival rate increases the response time follows the upper (minimum-speed) bound up to the point where its target utilization $U*$ is reached, then follows the descending curve for that $U*$ until it reaches the lower (high-speed) bound. The curves have the same zig-zag shape as those for the open system shown in Fig. 2(a).

The Markov model for this system follows that of Sect. 3 except for a modified arrival process with rate $(N - n)r$ arrivals/sec in queue state n. Some numerical solutions for four-level control, corresponding to the open arrivals case in Fig. 2, are displayed in Fig. 9. The results are very similar.



(a) $\Delta = 30$ sec (b) $\Delta = 100$ sec (c) $\Delta = 300$ sec

Fig. 9. Some speed-controlled response times for a closed M/M/1//N queue with N = 60

7 Conclusions

Speed-controlled processors can in some circumstances provide nearly constant response time over a wide range of throughputs, represented by the "plateau" approximation in Sect. 6. However with a power-saving strategy such as the "conservative" governor for

Linux, the response time increases steeply beyond this controlled range and performance collapses with little warning. If the power-management parameters are not ideal, pathological behaviour can set in, in the form of the non-monotonic (humped) response curves found for large normalized averaging times ($\mu\Delta$) and large target utilizations (U^*). The ideal settings for a particular workload may not be obtainable, or be stable over time, since they depend on the application-dependent CPU service times. Large target utilizations are common in practice and they make the plateau range smaller and the non-monotonic behaviour more severe.

These attributes of speed control raise challenges for performance management, particularly for autoscaling. Autoscalers are overwhelmingly based on an assumption that response time increases with increasing throughput, and the non-monotonic response curve could trap the node at a capacity well below what is achievable. Autoscaling based on a target utilization has less of this problem but could give reduced capacity because the speed control slows down the processor to raise the utilization (and save power).

These results also reveal challenges for performance modeling. The exact response time calculation is not tractable for practical solvers. The PK and plateau approximations may be useful but have limited accuracy in important cases. A known problem is the effect of speed control on the measurement of CPU demands for operations; the controller state must be known while measuring. Calibration of models from performance tests or from measurements made "in the wild" will be affected. This affects the usefulness of the models for capacity planning or deployment planning.

Most of these challenges can be offset by always using the "performance" governor, which uses minimum power with zero requests and switches immediately to maximum power when any task is scheduled. The response time is close to that for maximum power. This is the least power-efficient governor, but there seem to be compelling reasons to prefer it.

A natural conclusion is that only the "performance" governor has a useful future in performance-sensitive applications. In coming to this conclusion, the present research substantiates the usual recommendations regarding the choice of governor. It does not provide the best saving of power, suggesting that more stable controllers would be useful. The modeling approach taken here may be useful to evaluate other strategies.

Acknowledgements. Thanks to Wenbo Zhu for pointing out this problem where it arose in some measurements he made. This research was supported NSERC, the Natural Sciences and Engineering Research Council of Canada, by Discovery Grant RGPIN 06274-2016.

References

1. Anonymous: CPU governors explained, 27 June 2012. https://forum.xda-developers.com/showthread.php?t=1736168. Accessed 4 Jan 2020
2. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S.: Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications, Wiley (2006)
3. Brodowski, D.: Linux CPUFreq governors. https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt
4. Chen, T., Bahsoon, R., Yao, X.: A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems. ACM Comput. Surv. **51**(3), 1–40 (2018). Article 61

5. Doleželová, M., Heves, J., East, J., Domingo, D., Landmann, R., Reed, J.: Power management guide for RedHat enterprise Linux 6, section 3.2: using CPUFREQ GOVERNORS. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/power_management_guide/cpufreq_governors. Accessed 4 Jan 2020
6. Gough, C., Steiner, I., Saunders, W.: Energy efficient servers, chapter 2 CPU power management, pp. 21–70, and chapter 8 characterization and optimization, pp. 269–306, Apress (2015)
7. Jadoon, J.K.: Evaluation of power management strategies on actual multiprocessor platforms, Doctoral thesis, Universite de Nice – Sophia Antipolis, March 2013
8. Kidd, T.: Power management states: P-states, C-states, and package C-states, April 2014. http://software.intel.com/en-us/articles/power-management-states-p-states-c-states-and-package-c-states. Accessed Apr 2019
9. Li, B., León, E.A., Cameron, K.W.: COS: a parallel performance model for dynamic variations in processor speed, memory speed, and thread concurrency. In: Proceedings of HPDC 2017, the 26th International Symposium on High-Performance Parallel and Distributed Computing, Washington, pp. 155–166, June 2017
10. Lu, Y., Sharma, M., Squillante, M.S., Zhang, B.: Stochastic optimal dynamic control of Gi/Gi/1 queues with time-varying workloads. Probab. Eng. Inf. Sci. **30**, 470–491 (2016)
11. Microsoft: Processor power management options, 10 April 2017. https://docs.microsoft.com/en-us/windows-hardware/customize/power-settings/configure-processor-power-management-options. Accessed Apr 2019
12. Mutapcic, A., Boyd, S., Murali, S., Atienza, D., De Micheli, G., Gupta, R.: Processor speed control with thermal constraints. IEEE Trans. Circuits Syst. **56**(9), 1994–2008 (2009)
13. Rao, R., Vrudhula, S., Chakrabarti, C., Chang, N.: An optimal analytical solution for processor speed control with thermal constraints. In: Proceedings of International Symposium on Low Power Electronics and Design (ISLPED 2006), Tergensee, Germany, pp. 292–297, October 2006
14. Saber (psuedonym): Collective guide of CPU governors, I/O schedulers and other kernel variables, 8 March 2015. https://forum.xda-developers.com/t/ref-guide-most-up-to-date-guide-on-cpu-governors-i-o-schedulers-and-more.3048957/
15. Standard Performance Evaluation Corporation: SPECpower_ssj2008: Fujitsu FUJITSU Server PRIMERGY TX1320 M2, 25 November 2015. https://www.spec.org/power_ssj2008/results/res2015q4/power_ssj2008-20151110-00704.html. Accessed 7 Dec 2018