Markov Processes Relat. Fields **26**, 687–710 (2020)

Markov Processes and **Related Fields**

The SRPT Service Policy with Frequency Scaling: Modeling, Evaluation and Optimization

Andrea Marin[1] **, Isi Mitrani**[2] **, Maryam Elahi**[3] **and Carey Williamson**[4]

[1] Università Ca' Foscari Venezia, DAIS, Venice, Italy. E-mail: marin@unive.it
[2] Newcastle University, School of Computing, Newcastle, UK.
E-mail: isi.mitrani@newcastle.ac.uk
[3] Mount Royal University, Department of Mathematics and Computing, Calgary, Canada.
E-mail: melahi@mtroyal.ca
[4] University of Calgary, Department of Computer Science, Calgary, Canada.
E-mail: carey@ucalgary.ca

Received July 2, 2018, revised March 27, 2019

Abstract. We study a system where the speed of a processor depends on the current number of jobs. A queuing model in which jobs consist of a variable number of tasks, and priority is given to the job with the fewest remaining tasks, is analyzed in the steady state. The number of processor frequency levels determines the dimensionality of the queuing process. The objective is to evaluate the trade-offs between holding costs and energy costs, when setting the processor frequency. We obtain exact results for two and three frequency levels, and accurate approximations that can be generalized further. Numerical and simulation experiments validate the approximations and provide insights into the benefits to be gained from optimizing the system.

Keywords: queues, scheduling policies, speed control, energy saving, multidimensonal Markov processes, generating functions

AMS Subject Classification: 60J27

1. Introduction

In dynamic speed scaling systems, the speed at which the processor executes jobs is adjusted dynamically depending on the workload experienced by the system. Modern processors typically support over a dozen discrete operating

speeds, often with a factor of two (or more) between the slowest and the fastest speeds available.

Multiple tradeoffs exist in such speed scaling systems. The most obvious is the tradeoff between response time and energy consumption. To minimize response time, one would use the highest processor speed available, while to minimize energy consumption, one would use the lowest speed. For this reason, most speed scaling research uses a cost function that combines response time (i.e., job delay, or holding cost) and energy consumption when carrying out system optimization.

When the processor speed is set dynamically, depending on the current number of jobs in the system, the cost function is also affected by the job scheduling policy. Different policies produce different average numbers of jobs, and hence different costs.

The scheduling policy of interest to us is Shortest Remaining Processing Time (SRPT). It has been shown to minimize the system occupancy and hence the average response time, [11]. Also, it tends to make the CPU run for longer periods at lower speeds, which leads to low energy consumption. The downside of that policy is that it discriminates against long jobs, by making them wait longer and by serving them at lower speeds. However, despite concerns relating to fairness [1,3], the optimality of SRPT makes it an appealing choice in situations where job sizes can be predicted accurately. This is the case, for example, in transferring static resources from a web server, as shown in [8].

Our aim is to evaluate the tradeoffs involved in systems operating job-count speed scaling, in conjunction with SRPT scheduling. A typical question that arises in this context is: at what speed should the processor run when there is just one job present? High speed, to minimize delay, or low speed, to conserve energy? We intend to answer this, and other similar questions, by providing algorithms for computing the expected costs and searching for the optimal policy.

The main contributions of the paper are as follows. A queuing model of a system with a finite number, K, of processor speed levels is analyzed in the steady state. That number determines the dimensionality of the associated Markov process. Exact solutions are obtained in the special cases of $K = 2$ and $K = 3$. The computational complexity of those solutions is quite high, so simpler approximations are offered as efficient alternatives. The idea is to approximate the marginal probabilities of certain states by estimating the total amount of work done at the highest possible rate, per unit time in those states. These approximations may be generalized to larger values of K.

Numerical and simulation experiments are carried out, aimed at verifying the accuracy of the approximations, and providing new insights into the benefits of optimization. The advantages and disadvantages of the SRPT policy are also examined.

A part of this paper is presented in Quantitative Evaluation of SysTems (QEST'2018). The material that appears here, but not in the proceedings,

includes proofs of propositions, the exact and approximate solution of the model with three speed levels, and additional numerical experiments.

1.1. Related work

Prior literature on speed scaling systems appears in both the theory and systems communities. The theoretical work typically focuses on formal mathematical proofs of the properties of such systems, such as optimality and fairness. Systems research typically focuses on robust implementations. In this literature review, we focus primarily on the theoretical work, as relevant background context for our paper.

One of the first analytical studies of dynamic speed scaling systems in which jobs have explicit deadlines, and the service rate is unbounded, was caried out by Yao et al. [15]. An alternative approach that minimizes system response time, within a fixed energy budget, was considered by Bansal et al. [4]. George and Harrison [7] and Wierman et al. [13, 14] have focused on optimizing the speed scaling policy in the case where the queuing process is of the Birth-and-Death type (essentially an M/M/1 queue).

Energy-proportional speed scaling has been studied by Andrew et al. [1] and by Bansal et al. [2, 4]. In those studies, the cost function is not actually computed, but is characterized and bounded. In particular, those authors examine the 'competitive ratio' of the policy, i.e. the ratio between the cost of the given policy and the cost of an ideal optimal policy. In [2] it was shown that if $P(s)$ is the power consumption at speed s, and the speed used when there are n jobs present satisfies $P(s) = n + 1$, then SRPT is 3-competitive for an arbitrary function P. In a similar vein, [1] established that when the speed is set so that $P(s)$ is proportional to n, SRPT is optimal.

The issue of fairness in relation to the SRPT policy has been addressed, among others, by Bansal and Harchol-Balter [3], Wierman [12] and Andrew et al. [1]. The unfairness of SRPT may be aggravated by speed scaling. A policy such as Processor-Sharing (PS) is fair, but it is suboptimal in terms of response time and energy consumption [1].

Elahi and Williamson [6] have compared the policies PS and SRPT when the speed scaling function has the form $s = n^{1/\alpha}$, for some $\alpha \geq 2$. Analytical results were obtained in the case of PS, while SRPT was simulated. Different behaviours were observed, particularly under conditions of heavy load.

The distinguishing feature of the present work is the analysis of the SRPT policy under speed scaling, and the computation of the associated cost function.

The system model and its general properties are described in Section 2. The exact and approximate solutions for $K = 2$ are presented in Section 3 while Section 4 deals with the case $K = 3$. Section 5 presents the results of a number of numerical and simulation experiments. Some avenues for further research are outlined in Section 6.

2. Description of the model

Jobs arrive into the system in a Poisson stream with rate λ, and are served by a single server. Each job consists of a random non-empty batch of i.i.d. service phases which will be referred to as *tasks*. The duration of each task, if served at speed 1, is distributed exponentially with mean 1. The number of tasks in a job's batch will be referred to as the *size* of the job. Those sizes are i.i.d. random variables with an arbitrary distribution: a job has size i with probability q_i $(i = 1, 2, \ldots)$. The average job size, Q, is assumed to be finite.

This job composition means that the possible distributions of job *lengths* (i.e. the sums of their constituent tasks), belong to a large sub-class of the Coxian distributions (see [5]), which are known to be quite general for practical purposes.

The job scheduling policy is a version of SRPT based on remaining sizes, rather than lengths. That is, at any moment, the job with the smallest number of remaining tasks is served. That policy is combined with a control mechanism whereby the frequency of the processor, i.e. the speed at which it works, is scaled according to the current load. There are K possible frequency levels. If there is only 1 job present, it is served at rate μ_1 tasks per unit time; if there are 2 jobs, then the one with fewer remaining tasks is served at rate μ_2 tasks per unit time, with $\mu_2 > \mu_1$; ...; if there are K or more jobs present, then the one with the fewest remaining tasks is served at rate μ_K tasks per unit time, with $\mu_K > \mu_{K-1}$.

When the processor runs at speed μ_k, it consumes energy at a rate proportional to μ_k^α, where α is a constant which depends on the design of the processor; its value is usually between 1 and 3 (e.g., see [14]). To examine the trade-offs between holding costs and energy costs, we define a cost function, C, which is a linear combination of the two:

$$C = c_1 L + c_2 \sum_{k=1}^{K} u_k \mu_k^\alpha \ . \tag{2.1}$$

Here L is the average total number of tasks present in the system, c_1 and c_2 are given coefficients, and u_k is the probability that frequency level k is in operation. We shall assume, for simplicity, that an idle processor runs at speed μ_1. It would be just as easy to assume a different idling speed, or 0. The value of α does not affect the model; in all examples we shall take $\alpha = 2$.

The purpose of the subsequent analysis is to provide algorithms for computing L and u_k, and hence evaluate the cost function for a given set of parameters. The optimal values of μ_k can then be found by applying an appropriate numerical search.

The operation of the scheduling policy can be modeled by using K task queues, numbered 1, 2, ..., K. Sort the jobs present in the system in decreasing order of the numbers of their remaining tasks. Then queue 1 contains the

remaining tasks of job 1 (the largest in the system), queue 2 contains the remaining tasks of job 2, if any, ..., queue $K-1$ contains the remaining tasks of job $K-1$, if any, and queue K contains the remaining tasks of all other jobs, if any, whose sizes are smaller than that in queue $K-1$. Queue K is the only one where there may be tasks from more than one job. So, the number of tasks in queue i is always larger than or equal to the number in queue j if $i < j$ and $1 \leq i, j \leq K-1$. Queue K can contain an arbitrary number of tasks.

The state of the system at a given moment in time is a vector $(n_1,n_2,\ldots,n_K)$, specifying the contents of the K queues. That vector satisfies $(n_1 \geq n_2 \geq \cdots \geq n_{K-1})$, but it is possible that $n_K > n_{K-1}$. The server always serves the non-empty queue with the largest index, and if that index is i, it works at the rate of μ_i tasks per unit time.

An incoming job of size s tasks which finds the system in state $(n_1,n_2,\ldots,n_K)$ may cause a reassignment of tasks to queues, in order to preserve the shortest-remaining order. If $s \leq n_{K-1}$, then no such reassignment is necessary and the resulting state is $(n_1, \ldots, n_{K-1}, n_K + s)$. Otherwise, the incoming s tasks replace the content of queue i, where i is the lowest index such that $s > n_i$. Queue K receives the tasks from queue $(K-1)$, so that the new state is $(n_1, \ldots, n_{i-1}, s, n_i, \ldots, n_K + n_{K-1})$. If $s > n_1$, then the part of the vector preceding s is empty.

Consider now the steady-state probability, $p_k(n)$, that the total number of tasks in queues 1, 2, ..., k is n, while queues $k+1, \ldots, K$ are empty $(k = 1, 2, \ldots, K)$. For every k, $p_k(0)$ is the probability of an empty system, so we may sometimes omit the index and just write $p(0)$. The difference $p_k(n) - p_{k-1}(n)$, for $k = 1, 2, \ldots, K$ and $n > 0$, with $p_0(n) = 0$ by definition, is the probability that there are n tasks present and the non-empty queue with the highest index is queue k. In other words, that is the probability that there are n tasks present and the service rate is μ_k. These probabilities are 0 when $n < k$.

Let $w_k(z)$ be the generating function of $p_k(n)$:

$$w_k(z) = \sum_{n=0}^{\infty} z^n p_k(n) \ \ ; \ \ k = 1, \ldots, K \ .$$

The last of these functions, $w_K(z)$, corresponds to the distribution of the total number of tasks in the system. It satisfies the normalizing condition $w_K(1) = 1$.

The marginal probabilities, u_k, that the server works at rate μ_k (they appear in the definition (2.1) of the cost function C), are given by

$$u_1 = w_1(1) \ \ ; \ \ u_k = w_k(1) - w_{k-1}(1) \ \ ; \ \ k = 2, \ldots, K \ .$$

Let $a(z)$ be the generating function of the job size distribution:

$$a(z) = \sum_{n=1}^{\infty} z^n q_n \ .$$

We shall also need the excess probabilities, r_n, that the size of a job is strictly greater than n, for $n = 0, 1, \ldots$ ($r_0 = 1$). The generating function of r_n, $b(z)$, is given by

$$b(z) = \sum_{n=0}^{\infty} z^n r_n = 1 + \sum_{n=1}^{\infty} z^n \left[1 - \sum_{j=1}^{n} q_j\right] .$$

Table 1 summarizes the notations used in the paper.

λ	Arrival rate
q_i	$P(incoming\ job\ size = i)$
Q	Average job size
K	Number of frequency levels
μ_k	Service rate at level k
u_k	$P(service\ level = k)$
L	$E(total\ number\ of\ tasks)$
n_k	Number of tasks in queue k
$p_k(n)$	$P(n\ tasks\ in\ queues\ 1, \ldots, k;\ other\ queues\ empty)$
$\pi(n_1, \ldots, n_K)$	Joint stationary queue size distribution
$w_k(z)$	Generating function of $p_k(n)$
$a(z)$	Generating function of q_i
r_i	$P(job\ size > i)$
$b(z)$	Generating function of r_i

Table 1. Summary of notations

The following relation exists between $b(z)$ and $a(z)$:

$$b(z) = \frac{1 - a(z)}{1 - z} . \tag{2.2}$$

This is established by expanding $b(z) - zb(z)$ and performing cancellations. Note that the value of $b(z)$ at $z = 1$ is the average job size: $b(1) = a'(1) = Q$.

We have the following result.

Lemma 2.1. *The generating functions $w_1(z)$, $w_2(z)$, ..., $w_K(z)$ satisfy the relation*

$$w_K(z)[\mu_K - \lambda z b(z))] = \mu_1 p(0) + \sum_{k=1}^{K-1} (\mu_{k+1} - \mu_k) w_k(z) . \tag{2.3}$$

Proof. Make a cut in the state diagram between the set of states in which the total number of tasks in the system does not exceed n, and the set of states where that number exceeds n. The 'upwards' flows across the cut are from states with j jobs present ($j \leq n$), when a job of size greater than $n - j$ arrives.

The 'downwards' flows across the cut are from states with $n+1$ tasks present, when one of them completes. Bearing in mind that the service rate is μ_i when queue i is the non-empty queue with the highest index, which in those states happens with probability $p_i(n+1) - p_{i-1}(n+1)$, we can write

$$\lambda \sum_{j=0}^{n} p_K(j) r_{n-j} = p_1(n+1)\mu_1 + \sum_{i=2}^{K} \mu_i [p_i(n+1) - p_{i-1}(n+1)] \, .$$

This can be rewritten more conveniently as:

$$\lambda \sum_{j=0}^{n} p_K(j) r_{n-j} = \mu_K p_K(n+1) + \sum_{i=1}^{K-1} (\mu_i - \mu_{i+1}) p_i(n+1) \, . \tag{2.4}$$

Multiplying (2.4) by z^n and summing over all n gives, after a little manipulation and remembering that $\pi_i(0) = \pi_j(0)$ for all i, j,

$$\lambda z b(z) w_K(z) = \mu_K w_K(z) - \mu_1 p(0) + \sum_{i=1}^{K-1} (\mu_i - \mu_{i+1}) w_i(z) \, .$$

This is clearly equivalent to (2.3). □

Setting $z = 1$ in (2.3) yields the normalization condition:

$$w_K(1) = \frac{\mu_1 p(0) + \sum_{i=1}^{K-1} (\mu_{i+1} - \mu_i) w_i(1)}{\mu_K - \lambda Q} = 1 \, . \tag{2.5}$$

The following proposition gives the necessary and sufficient conditions for the stability of the system.

Proposition 2.1. *The queuing system is stable if and only if*

$$\lambda Q < \mu_K \, . \tag{2.6}$$

Proof. The necessity of (2.6) follows from the fact that, if $p(0)$ and $w_i(1)$ are positive and $\mu_{i+1} \geq \mu_i$ for $i = 1, 2, \ldots, K-1$, then the denominator in the right-hand side of (2.5) must be positive.

To prove the sufficiency, note first that the task arrival rate into queue K is bounded by λQ. Hence, condition (2.6) ensures that queue K is stable and, in particular, its average busy period, B_K, is finite.

Now consider queue $K-1$. The average interval that it takes to serve a task in it, S_{K-1}, is finite. Indeed, it is bounded by $(1 + \lambda B_K)/\mu_{K-1}$, since the average number of service interruptions does not exceed λ/μ_{K-1}, and the

duration of each interruption is B_K. On the other hand, when there are j tasks in queue $K-1$, the task arrival rate is bounded by

$$\gamma_j = \lambda \sum_{k=j+1}^{\infty} kq_k \ .$$

Since the average job size is finite, for every positive ϵ there is an integer J such that $\gamma_j < \epsilon$ when $j > J$. In particular, there is an integer J such that $\gamma_j < 1/S_{K-1}$, i.e. the drift of queue $K-1$ is negative, when $j > J$. When $j \leq J$, the drift is bounded. This implies, according to Pakes' lemma (see [9]), that queue $K-1$ is stable and its average busy period, B_{K-1}, is finite.

Repeating this argument for queues $K-2, \ldots, 1$, establishes the proposition. □

The steady-state average number of tasks in the system, L, is given by $w'_K(1)$. First, by differentiating (2.2) and applying L'Hôpital's rule at $z = 1$, we find that $b'(1) = a''(1)/2$. Now, taking the derivative of (2.3) at $z = 1$ and rearranging terms, we obtain

$$L = \frac{\sum_{i=1}^{K-1}(\mu_{i+1} - \mu_i)w'_i(1) + \lambda(Q + \frac{1}{2}a''(1))}{\mu_K - \lambda Q} \ . \tag{2.7}$$

Thus the quantities that are needed in order to evaluate the cost function C, given by (2.1), are expressed in terms of the functions $w_i(z)$ for $i = 1, 2, \ldots, K-1$, i.e. in terms of the probabilities of all states in which queue K is empty. In the following sections we provide exact and approximate solutions for those probabilities, in the cases $K = 2$ and $K = 3$. The methodology employed can be extended to deal with higher values of K, at the price of increased complexity.

The following general result will be useful. Denote by $\pi_1(i)$ the marginal probability that there are i tasks in queue 1 (and any numbers in the other queues). Then

$$\lambda r_n \sum_{i=0}^{n} \pi_1(i) = \mu_1 \pi(n+1, 0, \ldots, 0) \ , \tag{2.8}$$

where $\pi(n+1, 0, \ldots, 0)$ is the probability that queue 1 contains $n+1$ tasks and all other queues are empty. This equation is obtained by balancing the flows across a cut that separates the set of states with no more than n tasks in queue 1, from all other states.

The probabilities $\pi_1(i)$ can also provide an expression for the expected residence time, T_1, of a job in queue 1. Indeed, if the system is in state $(i, \cdot)$, then jobs arrive into queue 1 at rate λr_i. On the other hand, the average number of jobs in queue 1 (not tasks!) is equal to the probability that queue 1 is not empty, i.e., $1 - p(0)$. Therefore, according to Little's result:

$$T_1 = \frac{1 - p(0)}{\lambda \sum_{i=0}^{\infty} r_i \pi_1(i)} \ . \tag{2.9}$$

3. The model with $K = 2$ frequency levels

In this section we consider our model with two frequency levels, i.e., the server works at speed μ_1 when there is only one job in the system, and μ_2 when there are at least two jobs, with $\mu_1 < \mu_2$. In Section 3.1 we provide the exact solution of the model, whereas in Section 3.2 we give an approximate, yet a more efficient approach to the computation of the stationary performance indices. The accuracy of the approximation will be assessed in Section 5 and will be shown to be very high.

3.1. Exact analysis

The detailed system state is now a pair of non-negative integers, (i, j), where i is the number of tasks in queue 1 and j is the number of tasks in queue 2. The possible transitions out of state (i, j) are:

To state $(i-1, 0)$ with rate μ_1 if $i > 0$ and $j = 0$;

To state $(i, j-1)$ with rate μ_2 if $i > 0$ and $j > 0$;

To state $(i, j+k)$, $k = 1, 2, \ldots, i$, with rate λq_k;

To state $(k, j+i)$, $k = i+1, i+2, \ldots$, with rate λq_k.

The states $(0, j)$, for $j > 0$, are unreachable and their probabilities are 0.

Denote the probability of state (i, j) by $\pi(i, j)$. These probabilities satisfy the following global balance equations:

$$\lambda\pi(0,0) = \mu_1\pi(1,0)\ . \tag{3.1}$$

$$(\lambda+\mu_1)\pi(i,0) = \lambda q_i\pi(0,0) + \mu_1\pi(i+1,0) + \mu_2\pi(i,1)\ . \tag{3.2}$$

$$(\lambda+\mu_2)\pi(i,j) = \lambda\sum_{k=1}^{m} q_k\pi(i,j-k) + \lambda q_i\sum_{k=1}^{m_1}\pi(k,j-k) + \mu_2\pi(i,j+1)\,, \tag{3.3}$$

where $m = \min(i, j)$ and $m_1 = \min(i-1, j)$; $i > 0$, $j > 0$.

These homogeneous linear equations have infinitely many solutions which are all proportional to each other. If we find one solution and then normalize it by dividing each $\pi(i, j)$ by their sum, G, then the result will be the unique joint distribution of the two queue sizes.

Summary of the solution. The algorithm consists of a series of recurrent steps, carried out for increasing values of the number of tasks in queue 1. In step 1, the probabilities $\pi(1, j)$ are found by determining their generating function. In step i, for $i > 1$, the generating function of $\pi(i, j)$ is obtained in terms of those already determined.

Start by setting $\pi(0,0) = 1$. Equation (3.1) then gives $\pi(1,0) = \rho_1$, where $\rho_1 = \lambda/\mu_1$. Note that this quantity does not represent offered load, since λ is the arrival rate of jobs, while μ_1 is a service rate of tasks.

Consider the probabilities $\pi(1,j)$, for $j = 0, 1, \ldots$, and define the generating function

$$g_1(z) = \sum_{j=0}^{\infty} \pi(1,j) z^j \ .$$

When $i = 1$ and $j = 0$, equation (3.2) can be rewritten as

$$(\lambda + \mu_2)\pi(1,0) = \lambda q_1 \pi(0,0) + (\mu_2 - \mu_1)\pi(1,0) + \mu_1 \pi(2,0) + \mu_2 \pi(1,1) \ .$$

For $i = 1$ and $j \geq 1$, equations (3.3) are

$$(\lambda + \mu_2)\pi(1,j) = \lambda q_1 \pi(1,j-1) + \mu_2 \pi(1,j+1) \ .$$

Multiplying the above by z^j and summing, and remembering that $\pi(0,0) = 1$, we get

$$d_1(z) g_1(z) = \lambda q_1 z - [\mu_1 z + \mu_2(1-z)]\pi(1,0) + \mu_1 z \pi(2,0) \ , \qquad (3.4)$$

where

$$d_1(z) = \lambda z(1 - q_1 z) + \mu_2(z-1) \ .$$

This expression for $g_1(z)$ contains an unknown constant, $\pi(2,0)$. However, note that the coefficient $d_1(z)$ is negative for $z = 0$ and positive for $z = 1$. Therefore, there is a value, z_1, such that $d_1(z_1) = 0$. Since $g_1(z)$ is finite on the whole interval $z \in [0,1]$, the right-hand side of (3.4) must vanish at $z = z_1$. This gives

$$\mu_1 z_1 \pi(2,0) = [\mu_1 z_1 + \mu_2(1 - z_1)]\pi(1,0) - \lambda q_1 z_1 \pi(0,0) \ .$$

The next step is to consider the probabilities $\pi(2,j)$, for $j \geq 0$, and their corresponding generating function

$$g_2(z) = \sum_{j=0}^{\infty} \pi(2,j) z^j \ .$$

Repeating the manipulations that led to (3.4), we obtain

$$d_2(z) g_2(z) = \lambda q_2 z - [\mu_1 z + \mu_2(1-z)]\pi(2,0) + \mu_1 z \pi(3,0) + \lambda q_2 z^2 g_1(z) \ , \quad (3.5)$$

where

$$d_2(z) = \lambda z(1 - q_1 z - q_2 z^2) + \mu_2(z-1) \ .$$

Again, the coefficient of $g_2(z)$ is negative at $z = 0$ and positive at $z = 1$. Therefore, there is a value z_2 in the interval (0,1), such that $d_2(z_2) = 0$. Equating the

right-hand side of (3.5) to 0 at $z = z_2$, determines the single unknown constant in that equation, $\pi(3,0)$:

$$\mu_1 z_2 \pi(3,0) = [\mu_1 z_2 + \mu_2(1-z_2)]\pi(2,0) - \lambda q_2 z_2 \pi(0,0) - \lambda q_2 z_2^2 g_1(z_2) \ .$$

The i'th step in this process evaluates the generating function of the probabilities $\pi(i,j)$, $g_i(z)$, in terms of the already known functions $g_1(z)$, $g_2(z)$, ..., $g_{i-1}(z)$, and constants $\pi(1,0)$, $\pi(2,0)$, ..., $\pi(i,0)$:

$$d_i(z) g_i(z) = \lambda q_i z - [\mu_1 z + \mu_2(1-z)]\pi(i,0) + \mu_1 z \pi(i+1,0) + \lambda q_i z \sum_{k=1}^{i-1} z^k g_k(z) \ , \tag{3.6}$$

where

$$d_i(z) = \lambda z (1 - \sum_{k=1}^{i} q_k z^k) + \mu_2(z-1).$$

The coefficient of $d_i(z)$ has a zero, z_i, in the interval (0,1), which determines the new unknown constant $\pi(i+1,0)$.

These iterations continue until $g_i(1) < \epsilon$, for some sufficiently small ϵ, or until the largest possible value of i, if the job sizes are bounded. Eventual termination is guaranteed if the queuing process is stable. At that point, all (unnormalized) probabilities $\pi(i,0)$, and hence the function $w_1(z)$, have been determined.

The normalization constant, G, is given by (2.5):

$$G = \frac{\mu_1 \pi(0,0) + (\mu_2 - \mu_1) w_1(1)}{\mu_2 - \lambda Q} \ . \tag{3.7}$$

Dividing all $\pi(i,j)$ values, and hence $w_1(1)$, by G, completes the computation of the joint probability distribution of the states (i,j). Lemma 1 now provides $w_2(z)$.

The total average number of tasks in the system, L, is given by (2.7), which now has the form

$$L = \frac{(\mu_2 - \mu_1) w_1'(1) + \lambda[Q + \frac{1}{2} a''(1)]}{\mu_2 - \lambda Q} \ . \tag{3.8}$$

The probabilities that the processor speed is μ_1, or μ_2, are $w_1(1)$ and $1 - w_1(1)$, respectively.

Thus, the components of the cost function C are obtained in terms of

$$w_1(1) = \sum_i \pi(i,0) \ ,$$

and

$$w_1'(1) = \sum_i i \pi(i,0) \ .$$

3.2. Approximate solution

In order to derive the approximation, consider the marginal probabilities, $\pi(i,\cdot) = g_i(1)$, that there are i tasks in queue 1, with $\pi(0,\cdot) = \pi(0,0)$. These probabilities appear in the left-hand side of (2.8).

The fraction of time that the system spends in state $(i,\cdot)$, such that queue 2 is not empty, consists of the services of all tasks that join queue 2 when queue 1 reaches size i. This can happen when (a) the system is in state $(k,\cdot)$, for $1 \leq k < i$, and a job of size i arrives (in which case k tasks are transferred to queue 2), or (b) the system is in state $(i,\cdot)$ and a job of size k arrives, for $k \leq i$; all of its tasks join queue 2. Hence we can write

$$\pi(i,\cdot) - \pi(i,0) = \lambda \left[q_i \sum_{k=1}^{i-1} k\pi(k,\cdot) + \pi(i,\cdot) \sum_{k=1}^{i} kq_k \right] \frac{1}{\mu_2} \; ; \; i = 1, 2, \ldots \; . \quad (3.9)$$

The first sum in the right-hand side is absent when $i = 1$.

Introducing the notation

$$s_i = \sum_{k=1}^{i} k\pi(k,\cdot) \;\; ; \;\; a_i = \sum_{k=1}^{i} kq_k \; , \quad (3.10)$$

we can rewrite (3.9) as

$$\pi(i,\cdot) = \frac{\pi(i,0) + q_i \rho_2 s_{i-1}}{1 - \rho_2 a_i} \; ; \; i = 1, 2, \ldots \; , \quad (3.11)$$

where $\rho_2 = \lambda/\mu_2$ and $s_0 = 0$ by definition.

Start with $\pi(0,0) = 1$ and $\pi(1,0) = \rho_1$. Compute $\pi(1,\cdot)$ from (3.11), then $\pi(2,0)$ from (2.8), $\pi(2,\cdot)$ from (3.11), $\pi(3,0)$ from (2.8) and so on, up to the desired accuracy. Normalize, using (3.7).

This procedure is more economical and more stable than the exact solution. Moreover, it can be generalized to models with more than two queues.

The accuracy of the approximation will be examined in Section 5.

4. The model with $K = 3$ frequency levels

In this section we consider a system with three frequency levels, i.e., there are three service rates, $\mu_1 < \mu_2 < \mu_3$. As in Section 3 we provide an exact but computationally expensive analysis and an approximate but accurate and efficient solution.

4.1. Exact analysis

For $K = 3$, the system state is a triple, (i, j, m), indicating the numbers of tasks in queues 1, 2 and 3, respectively. Only the states for which $j \leq i$ are

feasible. Moreover, if $i = 0$, then $j = m = 0$; if $i > 0$ but $j = 0$, then $m = 0$. Denote by $\pi(i, j, m)$ the steady-state probability of state (i, j, m).

The arrival of a batch of size k, which occurs at rate λq_k, may cause the following transitions:

(a) From state $(0, 0, 0)$ to state $(k, 0, 0)$.

(b) From state $(i, 0, 0)$ $(i > 0)$ to state $(i, k, 0)$ if $k \leq i$, or to state $(k, i, 0)$ if $k > i$.

(c) From state (i, j, m) $(i, j > 0)$ to state $(i, j, m + k)$ if $k \leq j$, or to state $(i, k, m + j)$ if $j < k \leq i$, or to state $(k, i, m + j)$ if $k > i$.

The transitions caused by task completions are: from state $(i, 0, 0)$ $(i > 0)$ to state $(i - 1, 0, 0)$ at rate μ_1; from state $(i, j, 0)$ $(j > 0)$ to state $(i, j - 1, 0)$ at rate μ_2; and from state (i, j, m) $(m > 0)$ to state $(i, j, m - 1)$ at rate μ_3.

An exact solution of this model can be obtained by considering the generating functions

$$g_{i,j}(z) = \sum_{m=0}^{\infty} \pi(i, j, m) z^m \ \ ; \ \ i = 1, 2, \dots \ , \ j = 1, 2, \dots, i \ . \tag{4.1}$$

Start by setting $\pi(0, 0, 0) = 1$. Then a balance equation analogous to (3.1) gives $\pi(1, 0, 0) = \lambda/\mu_1$. The first of the generating functions (4.1), $g_{1,1}(z)$, is given by

$$d_1(z) g_{1,1}(z) = [\mu_3(z - 1) - \mu_2 z]\pi(1, 1, 0) + \lambda q_1 z \pi(1, 0, 0) \ , \tag{4.2}$$

where $d_1(z)$ has a similar form to the coefficient appearing in (3.4), but with μ_2 replaced by μ_3:

$$d_1(z) = \lambda z(1 - q_1 z) + \mu_3(z - 1) \ .$$

There is a value, z_1, in the interval (0,1), such that $d_1(z_1) = 0$. Equating the right-hand side of (4.2) to 0 at that point provides the probability $\pi(1, 1, 0)$. Then, equation (2.8) determines $\pi(2, 0, 0)$:

$$\mu_1 \pi(2, 0, 0) = \lambda r_1 [1 + \pi(1, 0, 0) + g_{1,1}(1)] \ .$$

In step i of this process, we have the following relations for $g_{i,1}(z)$, $g_{i,2}(z)$, ..., $g_{i,i}(z)$:

$$d_j(z) g_{i,j}(z) = x_{i,j}(z) + \lambda q_j z \sum_{k=1}^{j-1} g_{i,k}(z) z^k + 1(i > j) \lambda q_i z \sum_{k=1}^{j} g_{j,k}(z) z^k \ , \tag{4.3}$$

where $d_j(z)$ is the coefficient that appears in (3.6), but with μ_2 replaced by μ_3; $x_{i,j}(z)$ involves the boundary probabilities,

$$x_{i,j}(z) = \lambda q_j z\pi(i,0,0) + \lambda q_i z\pi(j,0,0) + [\mu_3(z-1) - \mu_2 z]\pi(i,j,0) + 1(i>j)\mu_2 z\pi(i,j+1,0) \; .$$

Note that the generating functions $g_{j,k}(z)$ for $j < i$ have already been determined, as well as the probabilities $\pi(i,0,0)$ and $\pi(j,0,0)$. The unknown quantities here are the i probabilities $\pi(i,j,0)$, and the i generating functions $g_{i,j}(z)$, for $j = 1, 2, \ldots, i$.

Each of the coefficients $d_j(z)$ has a zero in the interval (0,1). Equating the right-hand side of (4.3) to 0 at those points provides i simultaneous linear equations which can be used to compute the unknown probabilities, and hence the i generating functions. Equation (2.8) then determines $\pi(i+1,0,0)$.

At the termination of these iterations, all probabilities are divided by the normalization constant defined in (2.5).

4.2. Approximate solution

The implementation of the solution presented in Section 4.1 is quite complex and numerically expensive. It is therefore desirable to extend the approximation of the previous section to the case $K = 3$.

The idea is to approximate the marginal probability that there are i tasks in queue 1 and j tasks in queue 2, $\pi(i,j,\cdot)$, by estimating the total amount of work done at rate μ_3 per unit time in that state. The following events result in such work being contributed:

(a) A job of size $k \le j$ arrives and finds state $(i,j,\cdot)$.

(b) A job of size j arrives and finds state $(i,k,\cdot)$, with $k < j \le i$.

(c) A job of size i arrives and finds state $(j,k,\cdot)$, with $k \le j < i$.

In each of those cases, k tasks join, or are transferred to queue 3.

Introducing the notation

$$s_{i,j} = \sum_{k=1}^{j-1} k\pi(i,k,\cdot) \;\; ; \;\; t_{i,j} = 1(i>j) \sum_{k=1}^{j} k\pi(j,k,\cdot) \; ,$$

we can write an equation similar to (3.9)

$$\pi(i,j,\cdot) - \pi(i,j,0) = \lambda \left[\pi(i,j,\cdot) \sum_{k=1}^{j} q_k + q_j s_{i,j} + q_i t_{i,j} \right] \frac{1}{\mu_3} \; .$$

This yields a generalization of (3.11):

$$\pi(i,j,\cdot) = \frac{\pi(i,j,0) + \rho_3(q_j s_{i,j} + q_i t_{i,j})}{1 - \rho_3 a_j} \ , \tag{4.4}$$

where $\rho_3 = \lambda/\mu_3$ and a_j is given by (3.10).

In addition, the boundary probabilities $\pi(i,j,0)$ are approximated by using the recurrences

$$\mu_2 \pi(i,j+1,0) = \lambda r_j \left[\pi(i,0,0) + \sum_{k=1}^{j} \pi(i,k,\cdot) \right] . \tag{4.5}$$

These equations balance the flows across a cut between queue 2 states j and $j+1$, while ignoring all queue 1 states other than i.

Now, equations (4.4) and (4.5), together with (2.8), allow all required quantities to be computed in sequence, starting with $\pi(0,0,0) = 1$ and $\pi(1,0,0) = \rho_1$.

5. Numerical and simulation results

In this Section we describe several experiments aimed at evaluating the accuracy of the approximate solutions that have been proposed, and also observing the behaviour of the cost function (2.1). Systems with two and three frequency levels are examined. A remarkable observation emerging from these experiments is that, for the purposes of optimization, the models with $K > 2$ may be neglected.

$K = 2$ frequency levels

We consider the model studied in Section 3. In Figure 1, the cost function C is computed exactly and approximately, as described in section 3, and is plotted against the queue 1 service rate, μ_1. The two cost coefficients are $c_1 = 1$ and $c_2 = 2$. The job arrival rate and the queue 2 service rate are fixed at $\lambda = 1$ and $\mu_2 = 7$. A geometric distribution of job sizes is assumed, with mean 5, truncated at a maximum job size of 50. Thus the offered load, $\lambda Q/\mu_2$ represents about 71% utilization. The value of μ_1 is varied between 1 and 6, in increments of 1.

The cost function is convex in μ_1. We have no formal proof of this, but it is invariably observed to be the case. Intuitively, at low values of μ_1 the holding costs dominate, while at large values the service rate costs dominate. Moreover, if a point is reached such that an increase in μ_1 leads to an increase in C, then clearly any further increase in μ_1 would make matters worse.

The two plots are very close. The approximate solution underestimates C slightly, but the relative errors never exceed 1%. In particular, both solutions

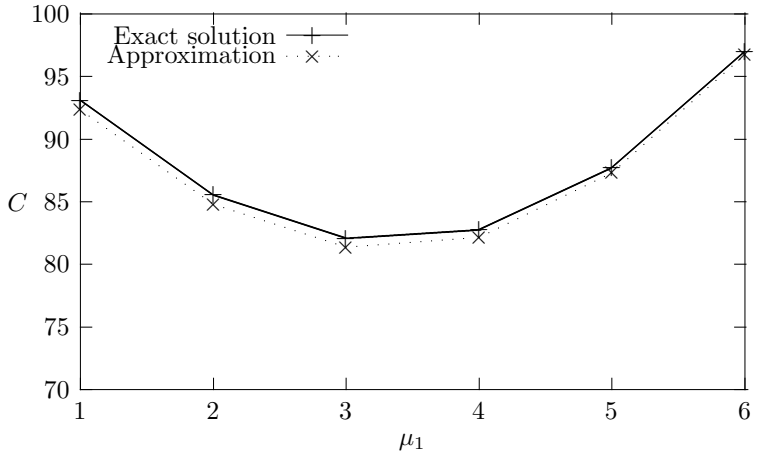

Figure 1. $K = 2$: Cost as a function of μ_1
$\lambda = 1$; $\mu_2 = 7$; $Q = 5$; $c_1 = 1$; $c_2 = 2$

agree that the optimal value of μ_1 is 3 (subject to the granularity of the increments).

We next examine the effect of increasing the variance of the job size distribution. Consider a rather extreme case where jobs have size 1, with probability 3/4, or size 17, with probability 1/4. The average job size is the same as in Figure 1, $Q = 5$, but the variance has gone up from 20 to 337. All other parameters are the same, and again the cost function is plotted against the service rate μ_1.

Figure 2 shows that the increase in variance has led to an increase in costs of between 7% and 10%. The approximate solution is still within less than 1% of the exact one. The optimal value of μ_1 has not changed.

The effect of the offered load on the shape of the cost function is illustrated in Figure 3. Three loading regimes are considered, light, moderate and heavy. These are represented by the arrival rates $\lambda = 0.3$, $\lambda = 0.8$ and $\lambda = 1.3$; they correspond to loadings of about 21%, 57% and 93%, respectively. The other parameters are the same as in Figure 1. Costs are evaluated exactly and are plotted against the queue 1 service rate, μ_1.

The figure suggests the following observations, all of which are quite intuitive. As the offered load increases, (a) costs increase; (b) the relative difference

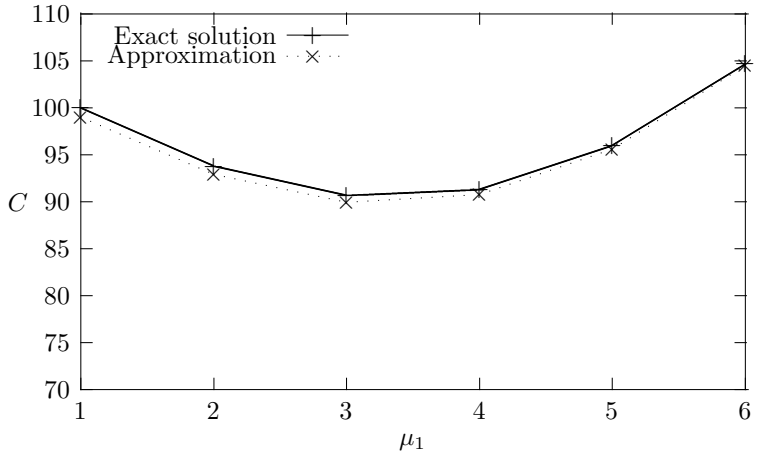

Figure 2. $K = 2$: Large variance of job sizes
$\lambda = 1$; $\mu_2 = 7$; $Q = 5$; $c_1 = 1$; $c_2 = 2$

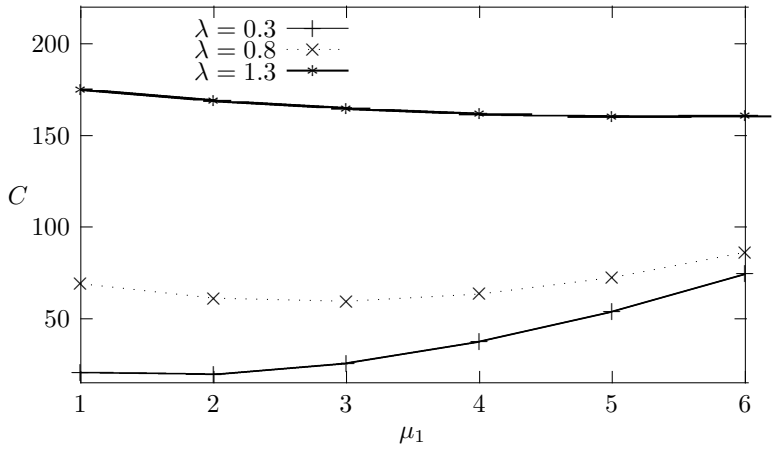

Figure 3. $K = 2$: The effect of load on costs
$\mu_2 = 7$; $Q = 5$; $c_1 = 1$; $c_2 = 2$

between the optimal and the pessimal costs decreases; (c) the optimal value of μ_1 increases.

$K = 3$ frequency levels

The next example evaluates the accuracy of the approximation proposed in Section 4, for a system with three frequency levels. This time μ_2 and μ_3 are fixed at 6 and 7 tasks per second respectively, while μ_1 is varied between 1 and 6, at increments of 1. The job size distribution is geometric with mean 5, and the other parameters are the same as before. Figure 4 compares the approximated costs with simulated ones. It was easier to simulate than to implement the exact solution. Each simulated point represents a run where $500,000$ jobs arrive into the system (i.e., an average of 2.5 million tasks are served).

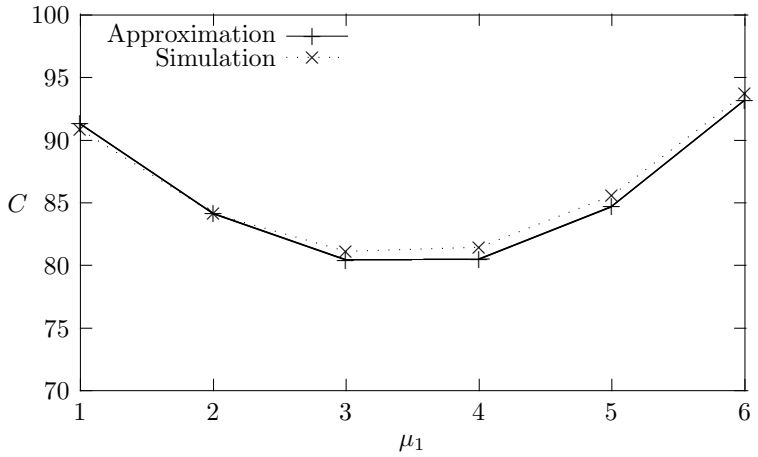

Figure 4. $K = 3$: Cost as a function of μ_1
$\lambda = 1$; $\mu_2 = 6$; $\mu_3 = 7$; $Q = 5$; $c_1 = 1$; $c_2 = 2$

The approximation is again very accurate, with relative errors not exceeding 1%. Moreover, the two plots agree on the optimal point of $\mu_1 = 3$ (subject to the granularity of the increments). However, the difference in costs between $\mu_1 = 3$ $\mu_1 = 4$ is very slight.

5.1. The benefits of optimization

Our next aim is to quantify the gains of frequency scaling, in the context of a 3-queue system under different loading conditions. Three policies are compared. The 'default' policy, or policy 0, does not optimize; it serves all three queues at rate μ_3. Under policy 0, the system is equivalent to a single queue with batch arrivals. Policy 1 serves queues 2 and 3 at rate μ_3, but uses the optimal value for μ_1 (found by a one-dimensional search). This amounts to an optimized $K = 2$

system. Policy 2 serves queue 3 at rate μ_3, but uses the optimal values for μ_1 and μ_2 (found by a two-dimensional search).

For consistency, all costs in this experiment were evaluated by applying the 3-queue approximation of section 4. We have relied on the established accuracy of that approximation. With a little extra effort, policies 0 and 1 could have been evaluated exactly.

In Figure 5, the costs incurred by the above three policies are plotted against the job arrival rate λ. It varies between 0.2 and 1.2, while the top service rate remains fixed at $\mu_3 = 7$. Job sizes are distributed geometrically with mean 5, which means that the system loading varies between 14% and 86%. The cost coefficients are $c_1 = 1$ and $c_2 = 2$. When searching for the best μ_1 and μ_2 values, the latter were incremented in steps of 0.5.

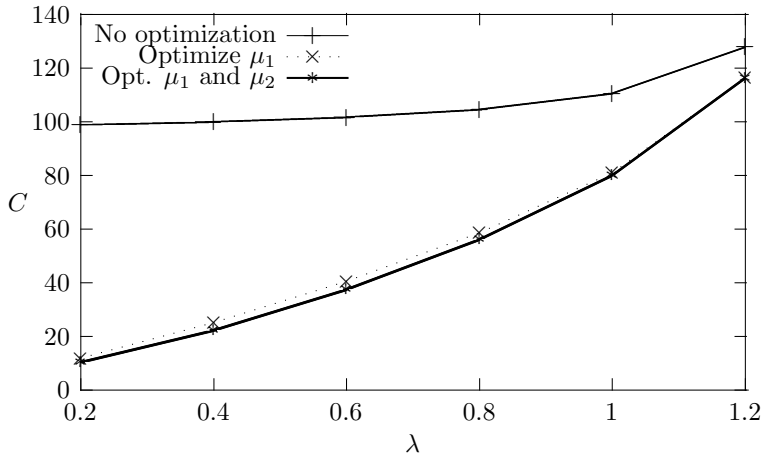

Figure 5. $K = 3$: The benefits of optimization
$\mu_3 = 7$; $Q = 5$; $c_1 = 1$; $c_2 = 2$

The results displayed in Figure 5 are quite instructive. First, we observe that there is less to be gained by optimizing a heavily loaded system, than a lightly loaded one. Of course this was to be expected, since the holding costs become dominant under heavy loads, and minimizing those costs requires large service rates.

The second observation is not so predictable: it seems that the big gains are obtained by optimizing just with respect to μ_1 (policy 1). The additional improvement achieved by optimizing with respect to μ_2 as well (policy 2), is quite minor. It is even debatable whether the expense of searching for policy 2 is justified by the benefits that it brings.

This last observation has practical importance. It suggests that the 2-queue model, rather than being just the simplest special case, is in fact a really significant model from the point of view of control and optimization. One may restrict the search for an optimal policy to the case $K = 2$, and be reasonably confident that the resulting policy would not be bettered by much.

To check whether the above conclusion remains valid under different cost structures, we have repeated the last experiment with several pairs of coefficients c_1 and c_2. Figure 6 shows one such example, where $c_1 = 2$ and $c_2 = 1$ (i.e., holding tasks in the system incurs higher penalties than speeding up the server). The other parameters remain unchanged, and the costs of policies 0, 1 and 2 are plotted against the arrival rate, λ.

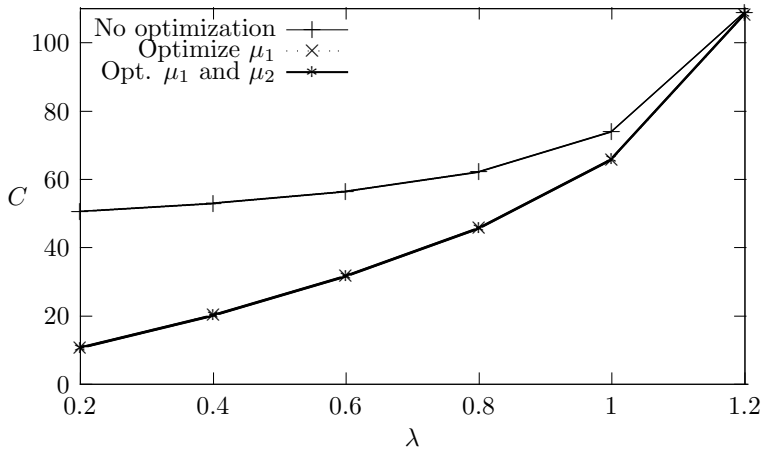

Figure 6. $K = 3$: Different cost coefficients
$\mu_3 = 7$; $Q = 5$; $c_1 = 2$; $c_2 = 1$

We observe again that policy 1 achieves significant gains compared to policy 0, but the additional benefits of optimizing with respect to μ_2 as well as μ_1 are negligible. The conclusion that the most important model is $K = 2$, is confirmed.

The last experiment attempts to compare the 'efficient but unfair' SRPT policy, with a 'fair but not so efficient' policy such as Processor-Sharing. A measure of the penalty inflicted by SRPT on long jobs is provided by the average residence, T_1, of a job in queue 1. That is the queue containing the job with the largest number of remaining tasks. We shall compare T_1 with the unconditional average response time, W_{SRPT}, of a job in the SRPT system, and also with the

response time, W_{PS}, of a job in a PS system with the same speed scaling. In addition, the cost of SRPT will be compared with that incurred by PS.

The Processor-Sharing queue with a K-level frequency scaling is a Birth-and-Death process with a constant arrival rate, λ, and a state-dependent service rate. In our case, the death rate is equal to μ_n/Q when there are n jobs present, for $n = 1, 2, \ldots, K-1$, and μ_K/Q when $n \geq K$ (Q is the average number of tasks per job). That process is easily solvable.

The following parameters are fixed: $K = 2$, $\mu_2 = 7$, $\beta = 0.8$ (i.e. $Q = 5$), $c_1 = 1$, $c_2 = 1$. The system loading is varied between about 20% and 80% by increasing λ. For each value of λ, the optimal μ_1 is found and the resulting SRPT and PS models are solved.

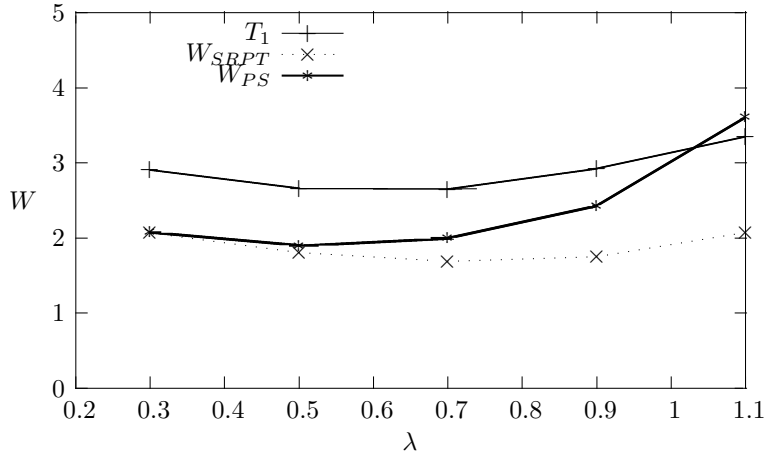

Figure 7. Average residence in queue 1, in SRPT system and in PS system
$K = 2$; $\mu_2 = 7$; $Q = 5$; $c_1 = 1$; $c_2 = 1$

In Figure 7, the values of T_1, W_{SRPT} and W_{PS} are plotted against λ. The first and third of these are computed exactly. The W_{SRPT} values were obtained by simulation, because our model provides performance measures for tasks, not jobs.

We observe that W_{SRPT} is lower than W_{PS}, and the difference between the two increases with the offered load. This is not surprising, since the SRPT policy is optimal. The penalty suffered by the longest jobs under SRPT is measured by the higher values of the conditional residence time, T_1. At heavy loads, however, the efficiency of SRPT causes even the long jobs to perform better than under PS.

Note that the curves are not monotone increasing. This is due to the optimization with respect to μ_1. For example, when λ increases from 0.3 to 0.4, the optimal μ_1 increases to an extent that actually reduces the residence times in both the PS and the SRPT queues.

Just for this example, in order to have a consistent comparison of costs between the two policies, we modify the cost function C by replacing the average number of tasks, L, in (2.1), with the average number of jobs present.

Figure 8 illustrates the costs achieved by the two policies. Over this range of loads, the costs of SRPT (which were again obtained by simulation), are lower than those of PS by about 10% — 13%.

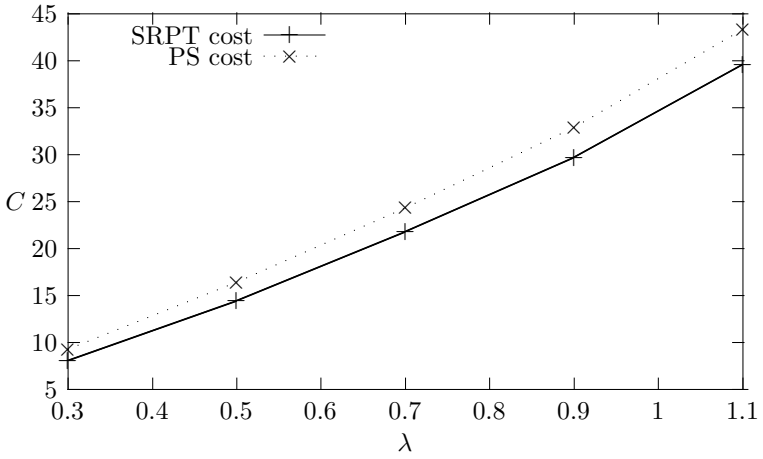

Figure 8. Costs under SRPT and PS policies
$K = 2$; $\mu_2 = 7$; $Q = 5$; $c_1 = 1$; $c_2 = 1$

One could summarize the last two figures by saying that the SRPT policy offers modest but significant efficiency gains, at the price of modest penalties to the longest jobs.

6. Conclusion

We have addressed the trade-offs arising in systems employing frequency scaling in conjunction with the Shortest Remaining Processing Time scheduling discipline. A non-trivial queuing model in which jobs represent batches of tasks was devised and analyzed. A cost function taking into account the system's power consumption and the expected number of tasks present was evaluated. Other metrics that were computed include the probabilities that the system

operates at its various frequency levels, the expected time that a job remains the largest job in the system and its expected size. Exact solutions were derived in the cases of 2 and 3 frequency levels, and accurate approximations were proposed. A number of numerical and simulation experiments provided important insights into the behaviour of the system. In particular, it seems that 2 frequency levels, properly optimized, are sufficient for practical purposes.

An interesting topic for future research would be to explore other scheduling policies that combine efficiency with a degree of fairness. One such candidate would be the Shortest Elapsed Time First (SETF) policy, see [10]. It has the advantage that the processing times or batch sizes do not need to be known in advance. However, the analytical challenges associated with that policy are considerable.

Acknowledgment

The work described in this paper has been partially supported by the Università Ca' Foscari Venezia - DAIS within the IRIDE program.

References

[1] L.L.H. Andrew, M. Lin and A. Wierman (2010) Optimality, Fairness, and Robustness in Speed Scaling Designs. *Proc. of ACM SIGMETRICS*, 37–48.

[2] N. Bansal, H. Chan and K. Pruhs (2013) Speed Scaling with an Arbitrary Power Function. *ACM Transactions on Algorithms* **9** (2), 18:1–14.

[3] N. Bansal and M. Harchol-Balter (2001) Analysis of SRPT scheduling: Investigating un-fairness. *Proc. of ACM SIGMETRICS*, 279–290.

[4] N. Bansal, T. Kimbrel and K. Pruhs (2007) Speed Scaling to Manage Energy and Temperature. *Journal of the ACM* **54** (1), article no. 3.

[5] D.R. Cox (1955) A use of complex probabilities in the theory of stochastic processes. *Mathematical Proceedings of the Cambridge Philosophical Society* **51** (2), 313–319.

[6] M. Elahi and C. Williamson (2016) Autoscaling Effects in Speed Scaling Systems. *Proc. of IEEE MASCOTS*, 307–312.

[7] J. George and J. Harrison (2001) Dynamic Control of a Queue with Adjustable Service Rate. *Operations Research* **49** (5), 720–731.

[8] M. Harchol-Balter, N. Bansal, B. Shroeder and M. Agrawal (2000) Implementation of SRPT scheduling in web servers. Technical Report CMU-CS-00-170.

[9] A.G. Pakes (1969) Some conditions for ergodicity and recurrence of Markov chains. *Operations Research* **17** (6), 1058–1061.

[10] I.A. Rai, G. Urvoy-Keller, M.K. Vernon and E.W. Biersack (2004) Performance analysis of LAS-based scheduling disciplines in a packet switched network. In: *Proc. Joint International Conference on Measurement and Modeling of Computer Systems* (*SIGMETRICS04/Performance04*), 106–117.

[11] L. Schrage (1968) A Proof of the Optimality of the Shortest Remaining Processing Time Discipline. *Operations Research* **16**, 678–690.

[12] A. Wierman (2011) Fairness and scheduling in single server queues. *Surveys in Operations Research and Management Science* **6** (1), 39–48.

[13] A. Wierman, L.L.H. Andrew and A. Tang (2009) Power-Aware Speed Scaling in Processor Sharing Systems. In: *Proc. of IEEE INFOCOM*, 2007–2015.

[14] A. Wierman, L.L.H. Andrew and A. Tang (2012) Power-Aware Speed Scaling in Processor Sharing Systems: Optimality and Robustness. *Performance Evaluation* **69**, 601–622.

[15] F. Yao, A. Demers and S. Shenker (1995) A Scheduling Model for Reduced CPU Energy. In: *Proc. of ACM Foundations of Computer Systems* (*FOCS*), 374–382.