# A Mixed PS-FCFS Policy for CPU Intensive Workloads

Simonetta Balsamo
Andrea Marin
Università Ca' Foscari Venezia
Venice, Italy
{balsamo,marin}@unive.it

Isi Mitrani
University of Newcastle
Newcastle, United Kingdom
isi.mitrani@newcastle.ac.uk

## ABSTRACT

Round robin (RR) is a widely adopted scheduling policy in modern computer systems. The scheduler handles the concurrency by alternating the run processes in such a way that they can use the processor continuously for at most a quantum of time. When the processor is assigned to another process, a context switch occurs. Although modern architectures handle context switches quite efficiently, the processes may incur in some indirect costs mainly due to cache overwriting.

RR is widely appreciated both in case of interactive and CPU intensive processes. In the latter case, with respect to the First-Come-First-Served approach (FCFS), RR does not penalise the small jobs.

In this paper, we study a scheduling policy, namely PS-FCFS, that fixes a maximum level of parallelism $N$ and leaves the remaining jobs in a FCFS queue. The idea is that of exploiting the advantages of RR without incurring in heavy slowdowns because of context switches.

We propose a queueing model for PS-FCFS allowing us to: (i) find the optimal level of multiprogramming and (ii) study important properties of this policy such as the mean performance measures and results about its sensitivity to the moments of the jobs' service demands.

## CCS CONCEPTS

• **Software and its engineering** → **Software performance**; *Scheduling*; • **Mathematics of computing** → *Queueing theory*.

## KEYWORDS

Scheduling, Response time optimisation, Queueing systems, Context switch

**ACM Reference Format:**
Simonetta Balsamo, Andrea Marin, and Isi Mitrani. 2022. A Mixed PS-FCFS Policy for CPU Intensive Workloads. In *Proceedings of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE '22), April 9–13, 2022, Bejing, China.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3489525.3511678


*ICPE '22, April 9–13, 2022, Bejing, China*

ACM ISBN 978-1-4503-9143-6/22/04…$15.00
https://doi.org/10.1145/3489525.3511678

## 1 INTRODUCTION

Processor sharing (PS) discipline has been widely studied in queueing theory especially motivated by computer science or telecommunication applications. In fact, the well-known round robin discipline with preemption of modern operating systems is well approximated by PS. In PS, each job in the queue receives the same amount of computational power and this approximates the fact that the round robin with preemption and resume implemented by operating systems assigns the processor to each of the ready jobs for a certain time-slice.

However, this approximation neglects the costs of the context switches. In fact, whenever a process is preempted and another one is resumed, the operating system must save the state of the former and restore the state of the latter. This causes what are known to be the *direct costs* of context switches and are estimated in modern architectures between 1 and 10 $\mu s$ according to the Intel-like architectures being considered (see, e.g., [14] and [26]) while, according to [4], in Linux on ARM architectures it raises to 48 $\mu s$.

For practical situations, direct costs may cause a small reduction in the performance of real world systems. However, context switches cause other time costs, namely the *indirect costs*. These mainly include the costs due to the sharing of L1-L3 cache systems among the parallel processes. Notice that, as for the Translation Lookaside Buffer (TLB) flushing penalties, these have been drastically reduced by modern processors both of AMD and Intel [18].

The indirect costs depend both on the architecture and on the application being run [4]. Several works agree on the fact that these penalties can reach up to few hundreds of microseconds per context switch [4, 14, 17].

In [9], the authors use SPEC CPU2006 benchmarks to assess the slowdown of processes for time-slices of 2.5 or 5ms. The measured slowdown can reach up to 50% for certain applications (hmme) with the smallest time-slice.

At this point, while it is clear that round-robin is useful for interactive processes, it becomes more difficult to say if this is a useful scheduling in the case of CPU intensive processes. In these cases, First-Come-First-Served (FCFS) discipline would eliminate almost entirely direct and indirect costs of context switches. However, it is well-know from queueing theory that, if the service demands of jobs (or their length) have a high coefficient of variation, the performance of FCFS in terms of response time can be extremely poor (see [12] for the results of the M/G/1 queueing system). Intuitively, this is due to the fact that a small job arriving after a big one has to wait for the completion of the latter before beginning its service. For example, an inversion of the order of service would drastically reduce the response time for the small job with a small penalty for the big one.

Processor sharing largely solves this problem. It is well-known that, under a Poisson arrival process, the expected response time depends only on the average service demand of the incoming jobs and not on its variance (or higher moments). The $M/G/1$ and the $M/G/1/PS$ queueing systems have the same expected response time when the service demand of the job has the variance of the exponential random variable. For higher coefficients of variation, the PS policy performs better than FCFS.

In the literature, a possible workaround to the penalty of the round robin is that of modulating the time-slice size dynamically. However, increasing the time-slice dynamically is in general complicated, and the decision algorithm itself may increase the overhead of the context switch (see, e.g., [21]). Moreover, longer time-slices may affect the performance of possible concurrent interactive processes and the control of the length is, in general, not possible in user mode.

In this paper, we propose a different approach to the problem. We schedule the jobs with a mixed discipline of PS and FCFS, i.e., at each moment at most $N$ jobs share the processor and the remaining ones are queued according to a FCFS policy. Figure 1 shows a graphical description of this discipline. Notice that the server has a state dependent speed to account for the slowdown caused by the PS. The slowdown function can depend on the number of concurrent jobs being concurrently served by the server. The goal is to enjoy the benefits of the PS discipline on the concurrent jobs but without paying a high penalty for the context switching.

Clearly, the new policy is useful only when the jobs' service demands have a variance higher than that of the exponential random variable, otherwise a plain FCFS would be simpler and would perform better. However, jobs' lengths are heavily tailed in practical situations as reported in many works (see, e.g., [6, 20] and the references therein).

This policy is not entirely new as it is used by some popular applications, as e.g., Apache web server. However, in these cases, the limitation of parallel processes is mainly done to contain the amount of resources globally used by the application. For example, for what concerns the memory resource, the objective is not to resort to page swapping that would lead to a drastic system slowdown.

The case of multi cores can be reduced to that of a single core by using the load-dependency of the service rate of the queueing system. However, in many scenarios, the possibility of specifying the affinity of a job to a core naturally reduces the problem of multi cores to that of finding the optimal level of concurrency for a single core [11, Ch. 35].

The research questions are twofold.

(i) One the one side, we aim to develop a model-based optimization procedure that allows us to find the optimal value for $N$ given the penalty of the context switches and service demand distribution. Clearly, if $N = 1$ the PS-FCFS policy corresponds to the simple FCFS while if $N \to \infty$ we obtain a state dependent PS queue. The PS-FCFS operates between these two limiting cases and, intuitively, finding the optimal $N$ to minimise the expected response time (or equivalently the average occupancy $L$) takes into account that, with FCFS, the server does not waste time in context switches but small jobs are penalised, while with a pure PS discipline the context switch costs negatively affect our goal.

(ii) The second problem that we address is more theoretical. It is well-known that under independent Poisson arrival process, the expected response time of the M/G/1/PS depends only on the first moment of the service demand, while for the FCFS discipline we must consider the first two moments. Our findings show that, surprisingly, the M/G/1/PS-FCFS queue is sensitive to moments higher than the second of the service demand. Although this observation has little practical consequences since, according to our experiments, the differences in the expected response times are small between distributions with the same first two moments, from a theoretical perspective we think it is an intriguing result.

The paper is structured as follow. Section 2 describes the motivations and the applications of the results of this research. In Section 3, we formally introduce the PS-FCFS discipline and introduce the modelling assumptions. In Section 4, we show its solution based on the generating function method. Section 5 discusses the sensitivity of the disciplines to the moments of the service time distribution. Section 6 presents some case studies and unveil some insights of the PS-FCFS discipline. Related work is discussed in Section 7. Finally, Section 8 concludes the paper.

## 2 MOTIVATIONS AND APPLICATIONS

Nowadays, parallel computations are extremely important especially in the domain of high performance computing. Hardware architectures provide several real or virtual cores each of which can run several processes or threads. While, in general, increasing parallelism is a clear advantage until the number of physical cores is reached (they generally have independent L1 and L2 cache systems and a very large L3 cache), it is less obvious how many processes should run for each core.

POSIX threads allow the programmer to specify the multiprogramming level for each core and the scheduling discipline that must be used to handle the parallelism [11, 22]: Round-Robin, First-Come-First-Served or *Other*, where the latter is the default one and uses the default policy of the operating system.

Linux scheduler program is called *Completely Fair Scheduler* (CFS) and decides the length of processes/threads' time-slice based on a dynamic priority that privileges interactive tasks and on the total number of processes. It is interesting to see that with higher number of parallel processes the time-slice is reduced and hence more context switches per unit of time are performed[1].

Our queueing model can be used, for example, in these scenarios:

(1) Assume we have a process with a certain (heavy) memory requirement and whose length is distributed according to a known distribution. For simplicity, let us ignore the context switch costs. What is the minimum level of parallelism (i.e., what is the minimum amount of memory that is needed) that we should use to enjoy the benefits of PS in the reduction of the expected response time?
(2) Assume a scheduling policy whose time-slice becomes smaller as the number of processes increases (e.g., the CFS). Given the job size distribution, the slowdown due to a context switch and the intensity of the workload, what is the optimal multiprogramming level decided considering the trade off between the reduction of the expected response time

[1] https://developer.ibm.com/tutorials/l-completely-fair-scheduler/

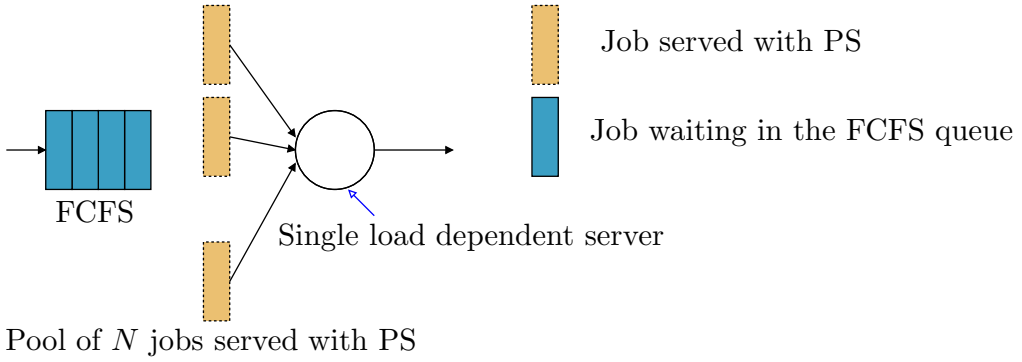



**Figure 1: The FCFS-PS policy**

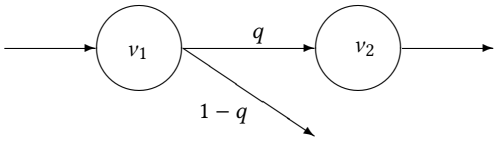



**Figure 2: Two-phase Coxian distribution**

and the penalty due to the context switch costs (direct and indirect)?

Questions similar to the second one require the estimation of the direct and indirect costs of context switches. There are several ways to do this, although the main methodology is the Ousterhout's method [19].

The answers to the previous questions depend, in general, on the traffic intensity. The numerical solution of the models is efficient and thus can be used for dynamical online configuration of the optimal maximum multiprogramming level.

## 3 THE FCFS-PS MODEL

Jobs arrive in a Poisson stream at rate $\lambda$ and are served by a single server with load-dependent speed. The job lengths (measured in instructions) are i.i.d. random variables with Coxian distribution with $K$ phases. Recall that Coxian distributions can model any distribution with rational Laplace transform and are dense in the domain of positive valued distributions.

Phase $i$ is distributed exponentially with mean $1/\nu_i$, for $i = 1, \ldots, K$. Phase $i+1$ follows phase $i$ with probability $q_i$. When the Coxian random variable consists of only 2 stages, we omit the subscript to $q_1$. Figure 2 shows an example of Cox-2 r.v..

The job scheduling policy is a combination of Processor-Sharing (PS) and First-Come-First-Served (FCFS), depending on a threshold parameter, $N$. That is, the jobs occupying the first $N$ positions in the queue share the processor equally, while the remaining jobs wait in order of arrival. Incoming jobs join the PS group if they find fewer than $N$ jobs present, otherwise they join the FCFS group. This policy will be referred to as FCFS-PS and is illustrated in Figure 1.

As previously discussed, the frequent context-switching inherent in the implementation of the PS discipline imposes a speed penalty which may depend on the number of jobs sharing, $n$. More precisely, the useful service capacity (measured in user instructions executed per unit time) is some non-increasing function of $n$, $s(n)$. For example, the service capacity could have the form $s(n) = a/(b+n)$, where $a$ and $b$ are positive constants. Thus, the processor works at its lowest service capacity, $s(N)$, when there are $N$ or more jobs present.

## 4 GENERATING FUNCTION SOLUTION FOR TWO PHASE COXIAN SERVICE DEMAND

In this section, we describe the solution for the case where the service time is distributed according to a Coxian r.v. with 2 phases. In Section 5, we consider the more general case of arbitrary number of phases.

Under this simplifying assumption, the system state is described by a pair of integers, $(n, i)$, where $n$ is the number of jobs present and $i$ is the number of jobs which are currently in phase 1 of their execution. The feasible states are $n = 0, 1, \ldots$ and $i = 0, 1, \ldots, \min(n, N)$. Since the variable $n$ increases and decreases by one job at a time, the above assumptions imply that $(n, i)$ is a Markov process of the Quasi-Birth-and-Death type. That process has a stationary distribution if the offered load (instructions per unit time) is lower than the lowest service capacity:

$$\lambda\left(\frac{1}{\nu_1} + \frac{q}{\nu_2}\right) < s(N)\ . \tag{1}$$

This condition is necessary and sufficient.

When the system is in state $(n, i)$, with $n < N$, a fraction $s(n)i/n$ of the service capacity is allocated to jobs in phase 1, and a fraction $s(n)(n-i)/n$ to jobs in phase 2. Hence, the completion rates for phases 1 and 2 are $s(n)i\nu_1/n$ and $s(n)(n-i)\nu_2/n$, respectively. To simplify the notation, we shall denote the phase 1 and phase 2 completion rates *per job* by $\mu_{1,n} = s(n)\nu_1/n$ and $\mu_{2,n} = s(n)\nu_2/n$, respectively. Then the overall phase 1 and phase 2 completion rates in state $(n, i)$, for $n < N$, can be expressed as $i\mu_{1,n}$ and $(n-i)\mu_{2,n}$, respectively.

If $n \geq N$, the completion rates cease to depend on $n$. We shall write $\mu_1 = s(N)\nu_1/N$ and $\mu_2 = s(N)\nu_2/N$, so that the overall phase 1 and phase 2 completion rates in state $(n, i)$ become $i\mu_1$ and $(N-i)\mu_2$, respectively.

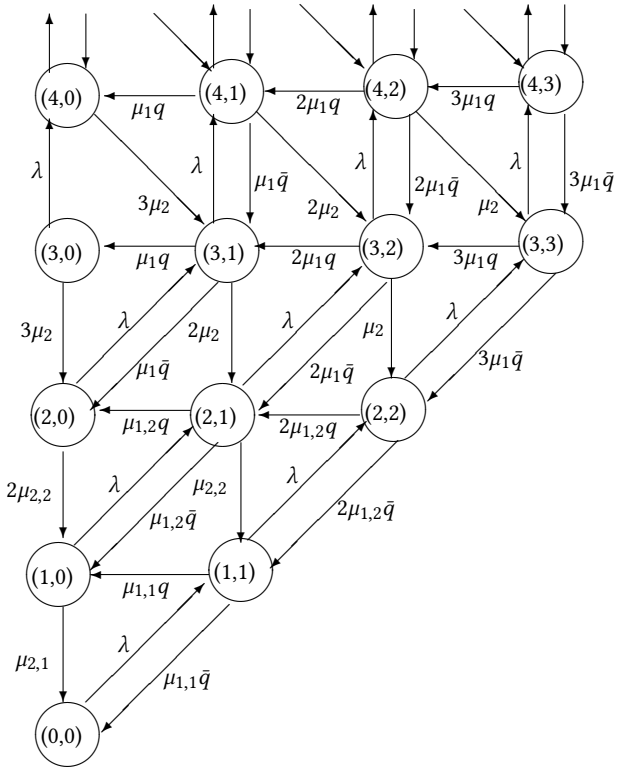



**Figure 3: State transition diagram for $N = 3$ ($\bar{q} = 1 - q$)**

The transition structure of the Markov process, for the case $N = 3$, is illustrated in Figure 3. Note that the state entered after a phase 1 completion is $(n, i-1)$ with probability $q$ (the job starts phase 2), and with probability $1-q$, it is either state $(n-1, i-1)$ (when $n \leq N$; the job departs and there are no waiting jobs), or state $(n-1, i)$ (when $n > N$; the job departs and is replaced by a waiting job in phase 1). A phase 2 completion results in state $(n-1, i)$ when $n \leq N$, and state $(n-1, i+1)$ when $n > N$ (a waiting job starts phase 1). A new arrival finding state $(n, i)$ causes a transition to state $(n+1, i+1)$ if $n < N$ and to state $(n+1, i)$ if $n \geq N$.

Let $\pi_{n,i}$ be the steady-state probability of state $(n, i)$. In view of the transition structure described above, we can write a set of balance equations for these probabilities. First, when $n = 0$,

$$\lambda\pi_{0,0} = \mu_{1,1}(1-q)\pi_{1,1} + \mu_{2,1}\pi_{1,0} \ . \tag{2}$$

For $n = 1, 2, \ldots, N-1$, the balance equations for $i = 0, 1, \ldots, n$ can be written as

$$(\lambda + i\mu_{1,n} + (n-i)\mu_{2,n})\pi_{n,i} = \lambda\pi_{n-1,i-1} + (i+1)\mu_{1,n}q\pi_{n,i+1}$$
$$+ (i+1)\mu_{1,n+1}(1-q)\pi_{n+1,i+1} + \mu_{2,n+1}(n+1-i)\pi_{n+1,i} \ , \tag{3}$$

where $\pi_{n-1,-1} = 0$ $\pi_{n,n+1} = 0$ by definition.

The balance equations for $n = N$ and $i = 0, 1, \ldots, N$ are

$$(\lambda + i\mu_1 + (N-i)\mu_2)\pi_{N,i} = \lambda\pi_{N-1,i-1} + (i+1)\mu_1 q\pi_{N,i+1}$$
$$+ i\mu_1(1-q)\pi_{N+1,i} + \mu_2(N-i)\mu_2\pi_{N+1,i-1} \ ; \ i = 0, 1, \ldots, N \ , \tag{4}$$

where $\pi_{N,N+1} = 0$ and $\pi_{N+1,-1} = 0$ by definition.

The balance equations for all states where $n > N$ have a slightly different first term in the right-hand side.

$$(\lambda + i\mu_1 + (N-i)\mu_2)\pi_{n,i} = \lambda\pi_{n-1,i} + (i+1)\mu_1 q\pi_{n,i+1}$$
$$+ i\mu_1(1-q)\pi_{n+1,i} + \mu_2(N-i)\mu_2\pi_{n+1,i-1} \ ; \ n > N, \ i = 0, 1, \ldots, N \ . \tag{5}$$

Again, $\pi_{n,N+1} = 0$ and $\pi_{n+1,-1} = 0$ by definition.

We will solve the infinite set of equations (4) and (5) by transforming them into a finite set involving $N+1$ generating functions defined as

$$g_i(z) = \sum_{n=N}^{\infty} \pi_{n,i} z^{n-N} \ \ ; \ \ i = 0, 1, \ldots, N \ . \tag{6}$$

Consider first the case $i = 0$. Multiplying equations (5) by $z^{n-N}$ and adding them to (4) produces the following equation,

$$[\lambda + N\mu_2]g_0(z) = \lambda z g_0(z) + \mu_1 q g_1(z) \ , \tag{7}$$

which we shall rewrite as

$$a_0(z)g_0(z) - \mu_1 q g_1(z) = 0 \ , \tag{8}$$

where

$$a_0(z) = \lambda(1-z) + N\mu_2 \ .$$

When $i > 0$, the resulting equations involve 'boundary' probabilities. Multiplying by $z^{n-N}$ and summing yields

$$[\lambda + i\mu_1 + (N-i)\mu_2]g_i(z) = \lambda\pi_{N-1,i-1} + \lambda z g_i(z) + (i+1)\mu_1 q g_{i+1}(z)$$
$$+ \frac{i\mu_1(1-q)}{z}[g_i(z) - \pi_{N,i}] + \frac{(N-i+1)\mu_2}{z}[g_{i-1}(z) - \pi_{N,i-1}] \ , \tag{9}$$

where the term involving $g_{i+1}(z)$ does not exist if $i = N$. This equation can be rewritten, after multiplying both sides by $z$ and rearranging terms, in the form

$$a_i(z)g_i(z) - (i+1)\mu_1 q z g_{i+1}(z) - (N-i+1)\mu_2 g_{i-1}(z) = b_i(z) \ , \tag{10}$$

where:

$$a_i(z) = \lambda z(1-z) + i\mu_1(z-1+q) + (N-i)\mu_2 z$$

for $i = 1, 2, \ldots, N$ and :

$$b_i(z) = \lambda z\pi_{N-1,i-1} - i\mu_1(1-q)\pi_{N,i} - (N-i+1)\mu_2\pi_{N,i-1}$$

for $i = 1, 2, \ldots, N$. Again, $g_{i+1}(z)$ appears only when $i < N$.

The above set of simultaneous equations can be written in matrix and vector form as follows.

$$A(z)\mathbf{g}(z) = \mathbf{b}(z) \ , \tag{11}$$

where $\mathbf{g}(z)$ is the column vector $[g_0(z), g_1(z), \ldots, g_N(z)]$, $\mathbf{b}(z)$ is the column vector $[0, b_1(z), \ldots, b_N(z)]$, and $A(z)$ is the tri-diagonal matrix

$$\begin{bmatrix} a_0(z) & -\mu_1 q & & & & \\ -N\mu_2 & a_1(z) & -2\mu_1 qz & & & \\ & -(N-1)\mu_2 & a_2(z) & -3\mu_1 qz & & \\ & & & \ddots & & \\ & & & -2\mu_2 & a_{N-1}(z) & -N\mu_1 qz \\ & & & & -\mu_2 & a_N(z) \end{bmatrix} .$$

The solution of (11) is given by

$$g_i(z) = \frac{D_i(z)}{D(z)} \ \ ; \ \ i = 0, 1, \ldots, N \ , \tag{12}$$

where $D(z)$ is the determinant of $A(z)$ and $D_i(z)$ is the determinant of the matrix obtained from $A(z)$ by replacing its $i+1$'st column with the column vector $\mathbf{b}(z)$ (it is customary to number the columns from 1 to $N+1$, rather than from 0 to $N$).

Thus all generating functions are determined in terms of the $2N+1$ unknown probabilities that appear in the elements of $\mathbf{b}(z)$: $\pi_{N-1,i}$, for $i = 0, 1, \ldots, N-1$ and $\pi_{N,i}$, for $i = 0, 1, \ldots, N$. One relation among these unknowns is obtained by setting $z = 0$ in (8):

$$(\lambda + N\mu_2)\pi_{N,0} - \mu_1 q \pi_{N,1} = 0 \ . \tag{13}$$

This coincides, of course, with (4) for $i = 0$.

Also unknown are the state probabilities that do not appear in $g_i(z)$: $\pi_{n,i}$, for $n = 0, 1, \ldots, N-2$ and $i = 0, 1, \ldots, n$. This makes a total of $(N+1)(N+2)/2$ unknown constants that need to be determined.

The balance equations (2) and (3), for $n = 0, 1, \ldots, N-1$ and $i = 0, 1, \ldots, n$, form $N(N+1)/2$ relations between the unknowns. These, together with (13), leave us needing a further $N$ equations, of which at least one must be non-homogeneous (i.e., have a non-zero right-hand side).

A set of $N-1$ additional equations are provided by the following result.

**Lemma**. *When the stability condition (1) holds, the polynomial $D(z)$ has exactly $N-1$ real and distinct roots, $z_1, z_2, \ldots, z_{N-1}$, in the open interval (0,1). There are no other roots in the interior of the unit disc.*

The proof of this Lemma is in the Appendix.

Consider one of the generating functions, say $g_0(z)$. Since it is finite on the interval (0,1), the numerator in the right-hand side of (12), $D_0(z)$, must vanish at each of the points $z_1, z_2, \ldots, z_{N-1}$, yielding $N-1$ equations. Using a different generating function would not provide new information.

The final, and only non-homogeneous equation, is the normalizing condition. All stationary probabilities $\pi_{n,i}$, for $n = 0, 1, \ldots$, and all feasible $i$, must sum up to 1. This can be written as

$$\sum_{n=0}^{N-1} \sum_{i=0}^{n} \pi_{n,i} + \sum_{i=0}^{N} g_i(1) = 1 \ . \tag{14}$$

The evaluation of $g_i(1)$, according to (12), is slightly complicated by the fact that both $D_i(z)$ and $D(z)$ are 0 when $z = 1$. One has to divide those two polynomials by $z-1$ before evaluating them at $z = 1$. That division can be carried out explicitly.

Replace the last row of $D(z)$ by the sum of all rows; this does not change its value. All elements of the new last row now have a common factor $z-1$, allowing us to write

$$D(z) = (z-1)\bar{D}(z) \ , \tag{15}$$

where $\bar{D}(z)$ differs from $D(z)$ only in the last row, which is now the row vector

$$\mathbf{d}(z) = [-\lambda, \mu_1 + (N-1)\mu_2 - \lambda z, 2\mu_1(1-q) + (N-2)\mu_2 - \lambda z, \ldots, N\mu_1(1-q) - \lambda z] \ . \tag{16}$$

A similar operation can be carried out with $D_i(z)$. Replacing its last row by the sum of all rows produces a common factor $z-1$ as before, except in the case of the $i+1$'st element, which is the sum of the elements of vector $\mathbf{b}(z)$ appearing in (10). That sum is given by

$$\sum_{i=1}^{N} b_i(z) = \lambda z \sum_{i=1}^{N} \pi_{N-1,i-1} - \sum_{i=1}^{N} [i\mu_1(1-q)\pi_{N,i} + (N-i+1)\mu_2 \pi_{N,i-1}] \ , \tag{17}$$

and can be rewritten as

$$\sum_{i=1}^{N} b_i(z) = (z-1)\lambda \sum_{i=1}^{N} \pi_{N-1,i-1} = (z-1)\lambda \pi_{N-1,\cdot} \ , \tag{18}$$

where $\pi_{N-1,\cdot}$ is the marginal probability that there are $N-1$ jobs present. Here we have used the equality

$$\lambda \sum_{i=1}^{N} \pi_{N-1,i-1} = \sum_{i=1}^{N} [i\mu_1(1-q)\pi_{N,i} + (N-i+1)\mu_2 \pi_{N,i-1}] \ , \tag{19}$$

which balances the flow from $N-1$ to $N$ jobs, with the flow from $N$ to $N-1$ jobs.

Thus we can write

$$D_i(z) = (z-1)\bar{D}_i(z) \ , \tag{20}$$

where $\bar{D}_i(z)$ is $D_i(z)$ with its last row replaced by $\mathbf{d}(z)$, and the $i+1$'st element of that row replaced by $\lambda \pi_{N-1,\cdot}$. The values of $g_i(z)$ at $z = 1$, in terms of the unknown probabilities, are obtained from

$$g_i(1) = \frac{\bar{D}_i(1)}{\bar{D}(1)} \ \ ; \ \ i = 0, 1, \ldots, N \ , \tag{21}$$

Having assembled and solved the set of equations determining the unknown probabilities, we can compute performance measures. In particular, the average number of jobs in the system, $L$, is given by

$$L = \sum_{n=1}^{N-1} n \sum_{i=0}^{n} \pi_{n,i} + \sum_{i=0}^{N} [g_i'(1) + N g_i(1)] \ , \tag{22}$$

The derivatives of $g_i(z)$ at $z = 1$ can be computed by following the rules for differentiating a determinant. However, a simpler way is to use the definition of a derivative:

$$g_i'(1) \approx \frac{g_i(1) - g_i(1-\delta)}{\delta} \ \ ; \ \ i = 0, 1, \ldots, N \ , \tag{23}$$

for some suitably small $\delta$.

## 5 SENSITIVITY ANALYSIS

In this section, we study the sensitivity of the expected performance indices to the moments of the job length distribution. The result that we show is rather counter intuitive although it mainly rises interesting theoretical observations rather than practical ones.

First, we recall that the M/G/1/PS queueing system enjoys the insensitivity property, i.e., the stationary distribution of the number of jobs in the queue (and hence all the expected stationary performance indices) depends only on the first moment of the job length distribution (see, e.g., [24]).

Conversely, the stationary distribution of the queueing system M/G/1/FCFS depends on the first two moments of the job length distribution and the expected response time is negatively affected by its coefficient of variation as derived by the famous P-K formula [12].

The scheduling discipline that we propose is a combination of PS and FCFS, hence one may intuitively guess that its expected

performance measures are independent of the moments of the job length distribution higher than the second. It is very surprising that this claim is *not* true.

This observation has theoretical interest but we argue that, for practical purposes, the analysis relying only on the first two moments is sufficient for at least two reasons. First, in many practical cases, the estimation of the moments higher than the second of the service demands may be difficult and many techniques rely on the first or on the first two moments (see, e.g., [8, 23] and [25] for an example of estimation of higher moments). Second, although we observe a discrepancy between the predicted performance indices of two distributions that share the first two moments but have different following ones, these are sufficiently small to make the use of only the first two moments accurate enough for practical purposes.

In order to study the sensitivity property of the FCFS-PS scheduling discipline, we use a matrix geometric solution based on the observation that we are considering a QBD process. With respect to the generating function method, this allows us to study Coxian distributions with more than two phases. The advantages of the generating function approach with respect to the QBD are discussed in Remark 1 at the end of this section.

## 5.1 Matrix geometric solution

The queueing process described in Section 3 can be seen as a QBD where 'levels' correspond to the number of jobs present in the FCFS queue.

In this section, we consider arbitrary $K > 0$ stages of service in the Coxian distribution.

Each level is described by the state of the jobs in service. We resort to the state representation that counts the number of jobs in service the are in each Coxian stage (see, e.g., [3]).

The state is described by the vector $(n_F, n_1, \ldots, n_K)$ where $n_F$ is the number of jobs queued in the FCFS buffer, $n_i$, $i = 1, \ldots, K$, is the number of jobs in the PS buffer being served at stage $i$. The first component of this state representation is the level of the QBD. When $n_F > 0$, an arrival with intensity $\lambda$ causes a passage from level $n_F$ to $n_F + 1$, while, if $n_F = 0$, the passage of level occurs only if $\sum_{i=1}^{K} n_i = N$, otherwise it causes an increment in the first component of the PS queue, $n_1$.

The service at stage $i$ in state $(n_F, n_1, \ldots, n_K)$ is given at rate $\mu_i s(n) n_i / n$, where $n = \sum_{j=1}^{K} n_j$.

When there is at least on job in the FCFS queue, the state of the PS part is a vector with $K$ components that sum to $N$. Thus, the number of possible states in the PS part when there is at least one job in the FCFS queue is:

$$D = \binom{N+K-1}{N}.$$

The infinitesimal generator has the following block-structure:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} & 0 & 0 & 0 & 0 & \cdots \\ \mathbf{B}_{10} & \mathbf{A}_1 & \mathbf{A}_2 & 0 & 0 & 0 & \cdots \\ 0 & \mathbf{A}_0 & \mathbf{A}_1 & \mathbf{A}_2 & 0 & 0 & \cdots \\ & & \ddots & \ddots & \ddots & & \end{pmatrix}. \tag{24}$$

Blocks $\mathbf{A}_0$, $\mathbf{A}_1$, $\mathbf{A}_2$ have size $D \times D$ and states inside are disposed in lexicographic order. Block $\mathbf{B}_{00}$ describes level 0 of the QBD, i.e., when there is not any job in the FCFS part. The number of PS jobs present ranges from 0 (the empty queue) to $N$. Taking into account the possible stage of service of each job, we have that the size of square matrix $\mathbf{B}_{00}$ is:

$$D_0 = \sum_{j=0}^{N} \binom{j+K-1}{j} = \frac{n+1}{k}\binom{k+n}{n+1}.$$

The sizes of $\mathbf{B}_{01}$ and $\mathbf{B}_{10}$ are $D \times D_0$ and $D_0 \times D$, respectively.

Let the rate matrix $\mathbf{R}$ be the solution of the equation:

$$\mathbf{A}_1 + \mathbf{R}\mathbf{A}_1 + \mathbf{R}^2\mathbf{A}_0\mathbf{A}_1^{-1} = \mathbf{0},$$

that can be numerically derived thanks to the logarithmic reduction algorithm [13].

Under stability conditions, the spectral radius of $\mathbf{R}$ is strictly lower than 1 and the following relation holds:

$$\boldsymbol{\pi}_i = \boldsymbol{\pi}_{i-1}\mathbf{R},$$

where $\boldsymbol{\pi}_i$ denotes the stationary probabilities of the states in level $i > 1$. The solution for levels 0 and 1 are found by solving the following linear system:

$$(\boldsymbol{\pi}_0, \boldsymbol{\pi}_1) \begin{pmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} \\ \mathbf{B}_{10} & \mathbf{A}_1 + \mathbf{R}\mathbf{A}_0 \end{pmatrix} = (\mathbf{0}, \mathbf{0}).$$

Since this system is homogeneous, it has to be normalised to ensure that the probabilities sum to 1, and the normalising constant is given by:

$$\alpha = |\boldsymbol{\pi}_0| + |\boldsymbol{\pi}_1(\mathbf{I} - \mathbf{R})^{-1}|,$$

where $\mathbf{I}$ is the identity matrix and $|\cdot|$ is the L1 norm operator.

Once we can compute the stationary distribution of the queue, we can obtain the expected number of jobs in the system in closed form. Let $\phi(\boldsymbol{\pi}_0)$ be the expected number of jobs in the systems when the FCFS part is empty, then:

$$L = \phi(\boldsymbol{\pi}_0) + (1 - |\boldsymbol{\pi}_0|)N + \boldsymbol{\pi}_1(\mathbf{I} - \mathbf{R})^{-2}. \tag{25}$$

The derivation of Equation (25) follows the standard methods of matrix geometric analyses.

Remark 1 (Comparison of the solution methods). *The generating function solution, when applicable, is exact and efficient. In the case of a 2-phase Coxian distribution, it requires the determination of $N-1$ roots in the interval (0,1) of a polynomial of degree $2N+1$, and then a solution of a set of $(N+1)(N+2)/2$ linear equations. That solution could be generalized to more than 2 phases, but the increase in complexity, due to both higher polynomial degree and larger set of equations, would be considerable. Moreover, when $N$ is large, some of the roots tend to bunch together, which causes the resulting matrix to become ill-conditioned. We have observed that, for the 2-phase Coxian, this happens when $N$ is larger than about 20.*

*The matrix-geometric solution is also, in principle, exact. It requires the determination of a $D \times D$ matrix $R$, and a solution of $D_0 + D$ linear equations. In practice, the computation of $R$ is approximate, since it involves an iterative algorithm. The numerical complexity of that computation increases quite steeply, not only with $D$, but also with the offered load. On the other hand, the matrix-geometric solution is easier to generalize to more than two phases. That is why we have adopted it for the evaluation of systems with 3-phase Coxian distribution.*

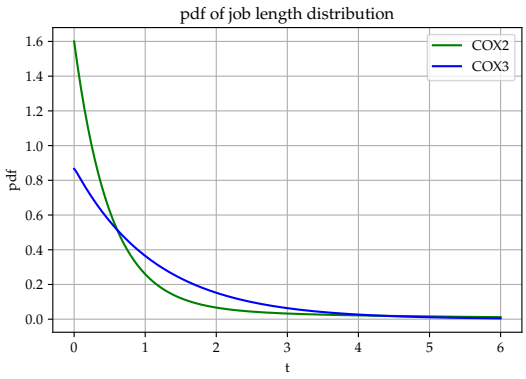



**Figure 4: pdfs of COX2 and COX3**

## 5.2 Sensitivity to the moments higher than the second of the job length distribution

In order to show that the PS-FCFS discipline is sensitive to moments higher than the second (in contrast with PS and FCFS) we consider two Coxian distributions and no state dependent service rate, i.e., $s(n) = 1$ for all $n > 0$.

The first, identified by COX2, consists of two phases and we have $q_1 = 0.2$ and $\nu_1 = 2$, $\nu_2 = 0.3$. The second, identified by COX3, consists of three phases with $q_1 = 8.97159 \cdot 10^{-3}$, $q_2 = 3.6498 \cdot 10^{-2}$ and $\nu_1 = 0.873685$, $\nu_2 = 56.656$, $\nu_3 = 0.0149258$. The probability density functions (pdfs) are plotted in Figure 4. The first and seconds moments of COX2 and COX3 are the same, i.e., $E[COX2] = E[COX3] = 1.16667$ and $E[COX2^2] = E[COX3^2] = 5.6111$. However, the other moments are, in general, different, e.g., $E[COX2^3] = 52.8611$ and $E[COX3^3] = 610.272$, $E[COX2^4] = 698.315$ and $E[COX3^4] = 161,179$.

Figure 5 shows the slowdown of the PS-FCFS discipline with respect to the simple PS. Since the coefficient of variation of the job size is larger than 1 and the PS is assumed not to give any speed penalty, obviously a higher parallelism brings benefits to the expected response time and PS-FCFS cannot have better performance than the simple PS.

Moreover, we observe that for $N = 1$ (purely FCFS system) and $N \to \infty$, COX2 and COX3 have the same slowdown with respect to PS since their first two moments are the same. The interesting part is that, when the two disciplines are mixed, they behave differently with COX3 enjoying slightly quicker that benefits of the combined discipline.

In moderate and heavy load conditions, the difference in the slowdown is below 10%.

# 6 CASE STUDIES

In this section, we propose three case studies. The first one aims at illustrating how the model can be used to estimate the optimal multiprogramming level when the penalty of context switches depends on the number of parallel processes.

The second and the third case studies use data retrieved from the literature to show two applications of the models. In the case study of Section 6.2, we assume a system scheduler in which the time slice is reduced proportionally to the number of ready processes. This is done to allow interactive processes to be more reactive with respect to the condition of a fixed time slice.

In the case study of Section 6.3, we work with a RR scheduler with fixed time-slice. With respect to the previous case, the slowdown due to the the context switches is the same for two or more ready processes.

## 6.1 Model driven optimisation of PS-FCFS

The first example is introduced to describe how the queueing model presented in Section 3 can be used to determine the optimal level of multiprogramming of the system when the time slice lenght is inversily proportional to the number of ready concurrent processes.

Recall that, the combined PS-FCFS policy is worth using only when the coefficient of variation of the job lengths is greater than 1. Otherwise, a straightforward FCFS policy, i.e. setting the threshold $N = 1$, is best. The question that arises in this connection is "for a given offered load and (high) cefficient of variation, how should one set the threshold $N$?"

To illustrate the optimisation procedure, we chose first an example of a 2-phase Coxian distribution where the average job length, $1/\nu_1 + q/\nu_2$, is 1. This involves no loss of generality. In this case, we set $\nu_1 = 2$, $\nu_2 = 0.4$ and $q = 0.2$. That is, phase 1 accounts for half of the average job length; phase 2 is 5 times longer but is present in only a fifth of the jobs. The second moment of the distribution, and hence the coefficient of variation, is $M_2 = 3.5$.

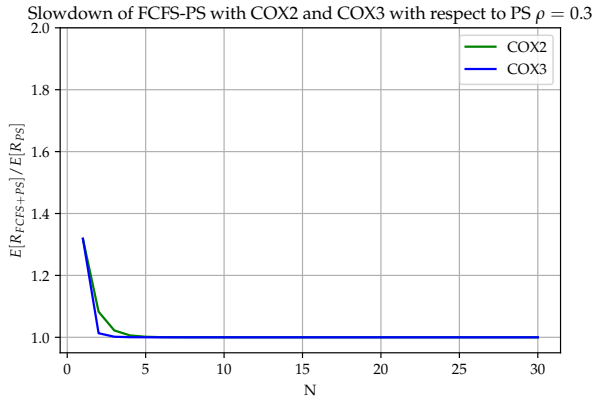
Suppose, to start with, that the useful service capacity does not depend on the number of jobs sharing the processor: $s(n) = 1$ for all $n$. In Figure 6a, the average number of jobs in the system is plotted against the threshold $N$, for two different arrival rates, $\lambda = 0.6$ and $\lambda = 0.8$. The results are obtained by applying the exact solution using the generating functions method.

It is to be expected that each plot should approach a horizontal asymptote corresponding to the performance of the pure PS policy. When the offered load is 0.6, that value is $L = 1.5$, while for the 0.8 load it is $L = 4$. What is less obvious is how quickly those asymptotes are approached. Under the moderate load, there is practically no difference in performance between $N = 8$ and $N = \infty$; even under the heavier load, it is enough to set $N = 15$ in order to get almost the full benefit of the PS policy.

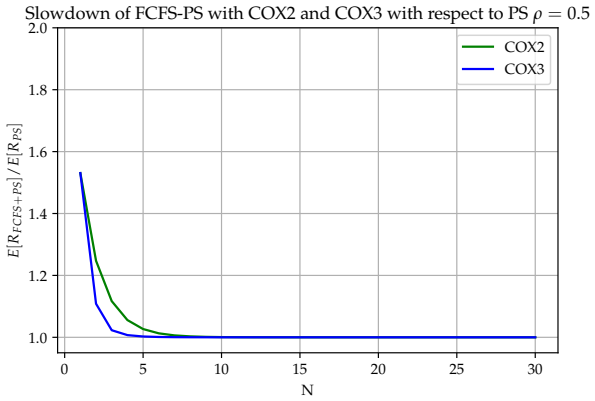
Now, consider the effect of speed penalties imposed by context switching. For the sake of this exposition, we shall assume that the useful service capacity decreases with the number of jobs sharing the processor according to function $s(n) = 30/(29 + n)$ (similarly to what is proposed in [10]). In other words, the speed is 1 for $n = 1$, and it drops by about 25% when $n = 10$.

We can expect a trade-off between the benefits of the PS policy and the loss of service capacity due to context switching. In general, there should be an optimal value of $N$. That trade-off is illustrated in Figure 6b, where $L$ is plotted against $N$ for an offered load of 0.6, and two different coefficients of variation. One as in the first example, and one where the second phase is 2 times larger and its frequency is 2 times smaller: $\nu_2 = 0.2$, $q = 0.1$. That increases the second moment to $M_2 = 6$.

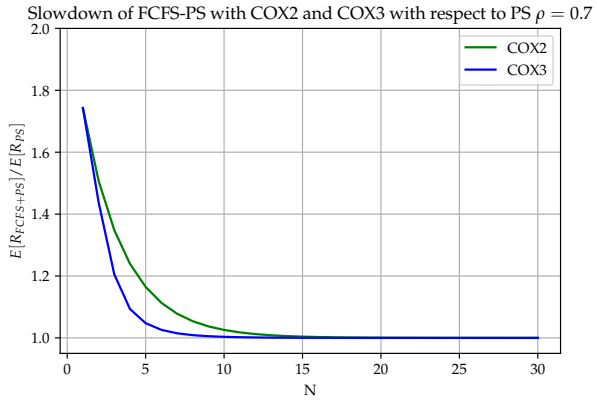
The optimal PS threshold is $N = 3$ when $M_2 = 3.5$, and $N = 6$ when $M_2 = 6$. Indeed, the larger the coefficient of variation, the



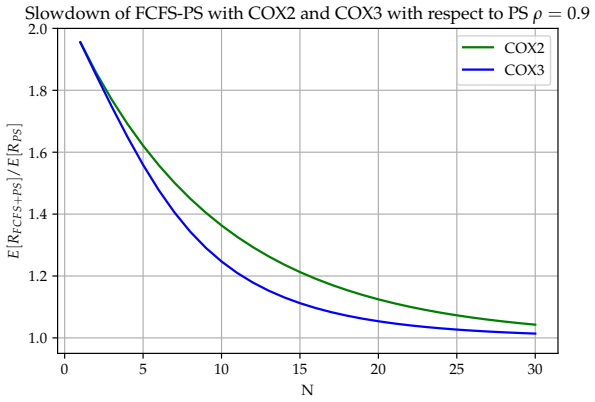

**(a)** $\rho = 0.3$




**(b)** $\rho = 0.5$

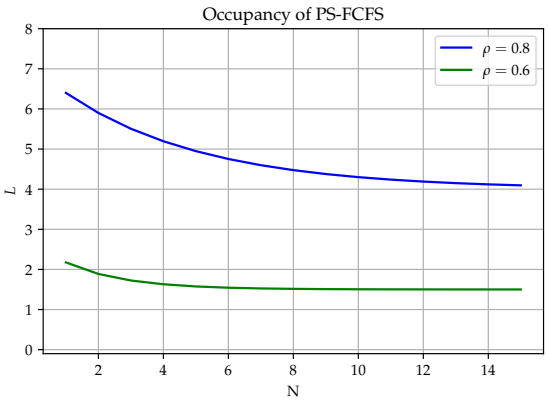



**(c)** $\rho = 0.7$

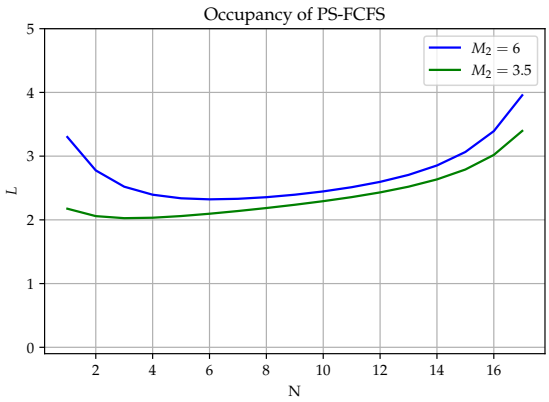



**(d)** $\rho = 0.9$

**Figure 5: Slowdown of COX2 and COX3 with respect to PS as function of the number of parallel servers**




**(a) Effect of PS threshold: no speed penalty**




**(b) Effect of PS threshold: decreasing service capacity**

**Figure 6: Behaviour of PS-FCFS with and without speed penalty**

bigger the incentive to use the PS policy. However, there always comes a point when the decrease in service capacity outweighs the benefits of PS. Moreover, the heavier the load, the sooner that point is reached.

## 6.2 Very heavily-tailed job sizes with scheduler with dynamic time-slice

In this case study, we use the measurements taken in [14]. According to the application type, the indirect costs of the context switches are evaluated from $2\mu s$ to one millisecond, according to the use of the cache. We adopt the results obtained from the linear scan of vectors which is a popular operation performed in many real world applications, i.e., we fix a context switch (indirect) cost of $500\mu s$.

More generally, we can give an explicit form of $s(n)$ as follows. To simplify the reasoning, we assume a cache system with only one level (e.g., consider only L3 cache). We call $n_0$ is the largest number of processes that can use the cache without serious trashing effects, and $n_1$ the largest number of jobs that fits in the main memory. Given a fixed time-slice $S$, we have:

$$s(n) = \begin{cases} 1 & \text{if } n \leq n_0 \\ \frac{S}{S+T_0} & \text{if } n_0 < n \leq n_1 \\ \frac{S}{S+T_1} & \text{if } n > n_1 \end{cases},$$

where $T_0$ and $T_1$ are the costs of restoring one process' cache or memory space from the disk, respectively. If $S = a/n$, then we have:

$$s(n) = \begin{cases} 1 & \text{if } n \leq n_0 \\ \frac{a}{a+nT_0} & \text{if } n_0 < n \leq n_1 \\ \frac{a}{a+nT_1} & \text{if } n > n_1 \end{cases}, \tag{26}$$

The reasoning can be extended to account for more levels of cache. Notice that many scheduler implementations have a minimum time-slice to avoid extremely frequent context switches. However, since we will see that the optimum number of concurrent processes is quite small, we can abstract out these details.

Now, assume that $a = 50ms$, and that we want to avoid swapping of the main memory by design, (i.e., the level of multiprogramming will be kept small enough to accommodate all the processes in the main memory), then we have:

$$s(n) = \frac{50000}{49500 + 500n} = 1/(0.99 + 0.01n)\,.$$

For example, with 10 jobs in the system the speed of service is approximately 92% of that measured for a single thread.

The job service demand is assumed to follow a bounded Pareto distribution as widely studied in the literature (see, e.g, [6, 20] and the references therein) for the jobs processes in Unix workstations. Coherently with these measurements, we assume $\alpha = 1.2$ and jobs range from a service demand of 1$s$ to $10^6$s. Thus, the cdf of service demand is given by:

$$F(x) = \frac{1 - (x_m/x)^\alpha}{1 - (x_m/x_M)^\alpha}\,,\ x_m \leq x \leq x_M\,,$$

where $x_m = 1s$, $x_M = 10^6 s$ and $\alpha = 1.2$. The mean of this distribution is $\mu^{-1} = 5.62$s and the coefficient of variation is 54.7263. The coefficient of variation is much larger than that of the exponential distribution, so the PS-FCFS discipline is worth of investigation.

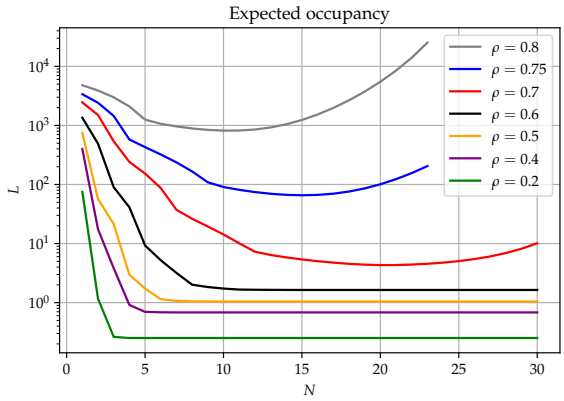



**Figure 7: Expected number of jobs in the queue with PS-FCFS policy. The offered load $\rho$ is computed with respect to the maximum speed of the server, i.e., as if the policy were FCFS, or $N = 1$. Y-axis is in logarithmic scale**

A simple fitting procedure gives the following values for the COX2 random variable:

$$q = 10^{-6}\,,\quad \mu_1 = 0.185s^{-1}\,,\quad \mu_2 = 4.60 \cdot 10^{-6}s^{-1}\,.$$

The very heavy tail of the distribution is evident from the low probability of accessing to the second phase of the Coxian random variable, and the very slow speed of this phase.

It is worth of notice that simulations of these types of distributions can be very time consuming since the effect of jobs accessing the second phase is very important in determining the mean performance indices but, on the other hands, these appear rarely in the arrival stream. As a consequence, very long simulations may be required to reach an accurate estimation of the stationary performance indices.

Figure 7 shows the expected number of jobs in the system as function of the maximum multiprogramming level, $N$ for several load factors. The load factor is computed as:

$$\rho = \frac{\lambda}{\mu \cdot s(1)}\,,$$

i.e., $\rho$ is the load factor of the FCFS policy, without considering the speed penalty.

The plot of Figure 7 shows several important properties of the PS-FCFS discipline. The first thing that we notice is that for low intensity workloads ($\rho = 0.2, 0.3$), a low level of multiprogramming, i.e., $N \simeq 5$, is sufficient to drastically reduce the expected response time (notice that the y-axis is in logarithmic scale). However, in these conditions, one may increase the level of multiprogramming without worrying too much about incurring the penalties of the context switches and making the PS discipline less efficient.

This is due to the fact that, even for large $N$, the low intensity workload makes the probability of observing many jobs in the system (and hence a strong reduction in the service speed) negligible.

With $\rho = 0.7$, the model shows that values of $N$ between 15 and 25 can be considered practically optimal for the minimization the

expected response time, as it becomes evident from the plateau of the curve.

The benefits for higher load factors, $\rho = 0.75$ and 0.8, are lower than in the previous cases, but still important. The plateau with the optimal values of $N$ becomes much narrower and high values of $N$ may bring the queue to instability.

These observations indicate that, in all cases, a relatively small value of $N$ would exploit the advantages of the PS policy. Moreover, the higher the load, the more important it is not to overestimate the threshold $N$.

In conclusion, this case study reveals an important insight of PS-FCFS policy: *in practice, the maximum level of multiprogramming should be low.*

Although, theoretically, the optimal value of $N$ decreases with growth of the offered load, in practice in low-load conditions, small values of $N$ are sufficient to enjoy most of the benefits of PS. This is because, even if we choose larger values for $N$, the small load factor makes very unlikely the event of observing many parallel jobs.

Optimal $N$ tends to be small even in heavy load. This is due to the fact that the slowdown due to the context switches becomes more important on heavy load as a consequence of the non-linearity of the variation of the expected response time with respect to the system load factor.

### 6.3 Fixed time-slice round-robin with memory limitation

Fixed time-slice RR is still available in Linux and Unix systems once the scheduling discipline for the process is set to *SCHED_RR*. This setting overrides the default policy and in general required administrator privileges to be set. The time-slice duration for *SCHED_RR* processes can be controlled by means of system setting.

In this case study, we assume a fixed time-slice RR and consider a job size distribution with lower variance than the previous, i.e.:

$$q = 10^{-4}, \quad \mu_1 = 0.185s^{-1}, \quad \mu_2 = 4.651 \cdot 10^{-4}.$$

In this case, we still have $\mu^{-1} = 5.62$s and the coefficient of variation is approximately $5.49 > 1$. As in the previous case, the context switch cost is set to $500\mu s$ while the fixed context switch time-slice is $5ms$. We can derive $s(n)$ easily:

$$s(n) = \begin{cases} 1 & \text{if } n = 1 \\ 5000/(5000 + 500) = 0.91 & \text{if } n > 1 \end{cases}.$$

The experiments with different offered load are shown in Figure 8. Both the plots show the penalty for $N = 2$ that is due to the cache trashing effect introduced by the RR. Since the context switch cost does not become more severe as the number of parallel processes grows, the curves asymptotically tend to the expected occupancy of a PS queue whose service rate is 91% of the nominal one.

Figure 8a shows that for moderate/high workloads (up to $\rho = 0.7$), a maximum multiprogramming level of $N = 10$ is sufficient to obtain the benefits of the PS discipline to reduce the expected response time. Compared to a pure PS policy, PS-FCFS ensures that the number of concurrent running processes is lower or equal to $N$. For example, if we assume $\rho = 0.7$, we see in Figure 8a that $N = 10$ is sufficient to drastically improve the expected response time. Thus, the main memory of the system should be sufficiently large to accommodate 10 processes. In contrast, if we use a pure PS scheduling, the probability of having more than 10 processes is $(0.7/0.91)^{11} \simeq 5.6\%$[2]. In other words, with the same amount of memory, we would have a non-negligible probability of observing page swapping that would drastically reduce the performance of the computation.

Very high workloads require higher levels of multiprogramming because the curve approaches the asymptotic line more slowly. This is depicted in Figure 8b. However, we should consider that a offered load of $\rho = 0.88$ implies a load factor perceived by the processor sharing of $0.88/0.91 \simeq 97\%$, i.e., we are handling an extreme case.

A final interesting observation regards the non-monotonicity of expected response time with respect to the maximum multiprogramming level $N$. Especially in heavy load, for small value of $N$, it may happen that the benefits of PS among the $N$ jobs are not sufficient to compensate the penalty of the speed reduction due to the indirect costs of the context switch. For example, for $\rho = 0.88$, even setting $N = 40$ is not sufficient to have a lower expected response time than the simple FCFS.

## 7 RELATED WORK

The cost of the context switch has drawn the attention of many researchers. From the application perspective, several authors have investigated the direct and indirect costs of this operation in various architectures and with various applications (see, e.g., [4, 14, 15]) following the approach described in the seminal work by Ousterhout [19].

Modern hardware architecture and the adoption of lightweighted threads make the direct costs of context switches very small [18]. However, their indirect costs are significant especially for certain workload types [1, 9].

In the literature, the low-level optimisation procedures that have been proposed to balance the advantages of the RR policy with its costs mainly focus on the definition of the appropriate time-slice (see e.g., [9, 21]). Long time-slices reduce the number of context switches but, on the other hand, they penalise interactive processes that may have to wait for the entire line of ready processes.

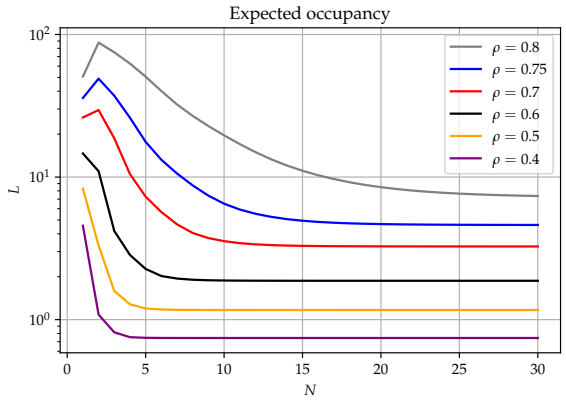
As a result of these and similar investigations, Linux CFS applies a combination of dynamic time-slice computation and priority assignment to handle the context switches.

From a higher level perspective, compilers are being designed to efficiently handle lightweight context switching in order to reduce the costs of parallel programming [5].

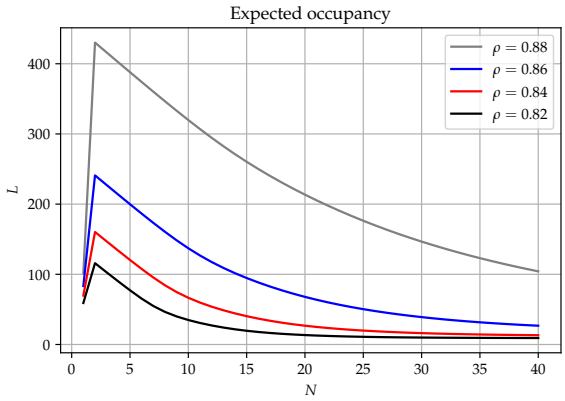
However, in our work, we study the problem at an even higher level. In fact, we do not aim to modify the compilers or the operating system configuration, but to provide a model to support the software developers in the decision of what is the optimal number of processes/threads that have to be launched in parallel in the system in order to minimise the response time or the hardware costs.

To the best of our knowledge, this is the first work that applies queueing theory to the problem of optimising the context switches, and the model that we propose is novel.

[2]This can be easily obtained, thanks to the insensitivity property, as the probability that a M/M/1 has more than 10 jobs in steady-state.




**(a) Expected number of jobs in the queue with PS-FCFS policy. The offered load $\rho$ is computed with respect to the maximum speed of the server, i.e., as if the policy were FCFS, or $N = 1$. Y-axis is in logarithmic scale**




**(b) Expected number of jobs in the queue with PS-FCFS policy. The offered load $\rho$ is computed with respect to the maximum speed of the server, i.e., as if the policy were FCFS, or $N = 1$. Y-axis is in linear scale**

**Figure 8: Behaviour of PS-FCFS with fixed time-slice RR**

Mixing of queueing disciplines has been previously studied in the context of age-based queueing policies (see., e.g., [2, 7, 16] and the references therein), but, in this work, we do not require to measure the age of the jobs.

## 8 CONCLUSION

The trade off between the benefits of the pseudo-parallel execution of threads and the costs of the context switches that they require has been known for a long time [19].

In this work, to the best of our knowledge, we provide the first quantitative model that studies the trade off between the expected response time reduction given by the PS policy with respect to FCFS in heavily tailed job sizes, and the slowdown caused by direct and indirect costs of context switches.

The scheduling discipline that we adopt and study consists in allowing a maximum multiprogramming level $N$ in a RR scheduler and store the remaining jobs in a FCFS queue.

The model allows the programmer to determine the optimal value of $N$ considering the expected response time and possibly the costs for the system set up.

Our investigation reveals that in most practical cases, a low value of $N$ (lower than 20) is sufficient to obtain the benefits of the PS discipline without wasting too much time in context switches or without requiring huge memory space to accommodate many parallel processes while avoiding memory swaps.

From the theoretical point of view, it is interesting to note that, while the M/G/1 and M/G/1/PS queues are sensitive to the first two and only first moments of the job size distribution, respectively, the performance of the M/G/1/PS-FCFS queue depends also on further moments.

However, for practical scenarios, the solution considering only the first two moments is an excellent approximation of the exact one, and allows the modeller to obtain an accurate estimation of the optimal value of $N$ in a computationally efficient way.

## REFERENCES


[1] A. Agarwal, J. Hennessy, and M. Horowitz. Cache performance of operating system and multiprogramming workloads. *ACM Trans. on Computer Systems*, 6(4):393–431, 1988.

[2] M. Akbari-Moghaddam and D. G. Down. SEH: size estimate hedging for single-server queues. In *Proc. of Quantitative Evaluation of Systems (QEST)*, pages 168–185, 2021.

[3] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. of ACM*, 22(2):248–260, 1975.

[4] F. M. David, J. C. Carlyle, and R. H. Campbell. Context switch overheads for Linux on ARM platforms. In *Proc. of the USENIX Workshop on Experimental Computer Science (ExpCS)*, page 3, 2007.

[5] S. Dolan, S. Muralidharan, and G. David. Compiler support for lightweight context switching. *ACM Trans. on Architecture and Code Optimization*, 9(4):1–25, 2013.

[6] M. Harchol-balter and A. B. Downey. Exploting process lifetime distributions for dynamic load balancing. *ACM Trans. on Computer Systems*, 15(3):253–285, 1996.

[7] E. Hyytiä and S. Aalto. On round-robin routing with FCFS and LCFS scheduling. *Perform. Evaluation*, 97:83–103, 2016.

[8] E. Incerto, A. Napolitano, and M. Trobastone. Learning queuing networks via linear optimization. In *Proc. of ACM/SPEC Int. Conf. on Performance Engineering (ICPE)*, pages 51–60, 2021.

[9] N. Jammula, Qureshi M, A. Gavrilovska, and J. Kim. Balancing context switch penalty and response time with elastic time slicing. In *Proc. of Int. Conf. on High Performance Computing*, pages 1–10, 2014.

[10] K. Kant. *Introduction to Computer System Performance Evaluation.* Mcgraw Hill Computer Science Series. Mcgraw Hill, 1992.

[11] M. Kerrisk. *The Linux programming interface. A Linux and Unix system programming handbook.* No Starch Press, 2010.

[12] A. Y. Khintchine. Mathematical theory of a stationary queue. *Matematicheskii Sbornik*, 39(4), 1932.

[13] G. Latouche and V. Ramaswami. A logarithmic reduction algorithm for quasi-birth-death processes. *J. of Applied Probability*, 30(3):650–674, 1993.

[14] C. Li, C. Ding, and K. Shen. Quantifying the cost of context switch. In *Proc. of the USENIX Workshop on Experimental Computer Science (ExpCS)*, pages 1–4, 2007.

[15] F. Liu, F. Guo, Y. Solohin, and A. Eker. Characterizing and modeling the behavior of context switch misses. In *Proc. of Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 91–101, 2008.

[16] A. Marin, S. Rossi, and C. Zen. Size-based scheduling for TCP flows: Implementation and performance evaluation. *Comput. Networks*, 183:107574, 2020.

[17] J. C. Mogul and A. Borg. The effect of context switches on cache performance. In *Proc. of the Forth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)*, pages 75–84, 1991.

[18] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. Intel virtualization technology: Hardware support for efficient processor virtualization. *Intel Technology*

*Journal*, 10(3), August, 2006.
[19] J. K. Ousterhout. Why aren't operating systems getting faster as fast as hardware? In *Proc. of the USENIX Summer Conference*, pages 247–256, 1990.
[20] K. Psounis, P. Molinero-Fernandez, B. Prabhakar, and F. Papadopoulos. Systems with multiple servers under heavy-tailed workloads. *Performance Evaluation*, 62:456–474, 2005.
[21] G. S. Rao, N. Srinivasu, S. V. N. Srinivasu, and G. R. K. Rao. Dynamic time slice calculation for round robin process scheduling using NOC. *Int. J. of Electrical and Computer Engineering (IJECE)*, 5(6):1480–1485, 2015.
[22] K. A. Robbins and S. Robbins. *UNIX Systems Programming*. Prentice-Hall, 2003.
[23] S. Spinner, G. Casale, F. Brosig, and S. Kounev. Evaluating approaches to resource demand estimation. *Performance Evaluation*, (92):51–71, 2015.
[24] P.G. Taylor. Insensitivity in stochastic models. In R. J. Boucherie and N. M. van Dijk, editors, *Queueing Networks: a Fundamental Approach*, pages 121–140. Springer, 2011.
[25] R. Wang, G. Casale, and A. Filieri. Service demand distribution for microservices using markovian arrival processes. In *Proc. of Int. Conf. on Quantitative Evaluation of Systems (QEST)*, pages 310–328, 2021.
[26] V. M. Weaver. Linux perf_event features and overhead. In *Proc. of FastPath Workshop*, page 7, 2013.

## APPENDIX

**Proof of Lemma.**

Consider the sequence of $N+1$ polynomials formed by the principal diagonal minors of the determinant $D(z)$. That is, define

$$Q_0(z) = a_0(z) \quad (27)$$

$$Q_1(z) = a_1(z)a_0(z) - \mu_1 q N \mu_2 \quad (28)$$

$$Q_i(z) = a_i(z)Q_{i-1}(z) - i\mu_1 qz(N-i+1)\mu_2 Q_{i-2}(z) \quad (29)$$
$$; \ i = 2, 3, \ldots, N \ .$$

The last polynomial in the sequence, $Q_N(z)$, is $D(z)$. The following properties hold.

(1) $Q_0(z) > 0$ in the interval (0,1).
(2) The degree of $Q_i(z)$ is $2i+1$, $i = 0, 1, \ldots, N$.
(3) If $Q_{i-1}(z) = 0$ for some $z$, then $Q_{i-2}(z)$ and $Q_i(z)$ have opposite signs at that point, $i = 2, 3, \ldots, N$. This follows from (29).

These properties imply that $Q_0(z)$, $Q_1(z)$, ..., $Q_N(z)$ is a generalised Sturm sequence for the polynomial $D(z)$. However, we cannot apply Sturm's theorem directly, because (a) $Q_N(1) = 0$, and (b) $Q_{N-1}(z)$ is not the derivative of $Q_N(z)$. Instead, we shall exploit the properties of the sequence.

Setting $z = 0$ in $Q_i(z)$ and using the definitions of $a_i(z)$ together with the recurrences (27), (28), (29), shows that $Q_0(0) = \lambda + N\mu_2$; $Q_1(0) = -\mu_1[\lambda(1-q) + N\mu_2]$; $Q_i(0) = -i\mu_1(1-q)Q_{i-1}(0)$, $i = 2, 3, \ldots, N$. That is, $sign[Q_i(0)] = (-1)^i$, $i = 0, 1, \ldots, N$.

Similarly, setting $z = 1$ in $Q_i(z)$ yields $Q_i(1) = N(N-1)\cdots(N-i)\mu_2^i$, for $i = 0, 1, \ldots, N$. Consequently, $sign[Q_i(1)] = 1$ for $i = 0, 1, \ldots, N-1$.

Starting with $Q_0(z)$, which is linear, we note that it is positive at $z = 1$ and negative at $z = \infty$. Hence, its single root, $z_{0,1}$, is greater than 1. Next, $Q_1(z)$, which is of degree 3, is negative at $z = 0$, positive at $z = 1$, negative at $z = z_{0,1}$ (according to (28)), and positive at $z = \infty$. Hence, it has three distinct real roots, $z_{1,1}$, $z_{1,2}$ and $z_{1,3}$, of which the first is between 0 and 1, the second is between 1 and $z_{0,1}$ and the third is between $z_{0,1}$ and $\infty$.

At those three points, the sign of $Q_2(z)$ is (-,-,+), respectively, according to (29). Hence, the degree 5 polynomial $Q_2(z)$ has a real root in each of the intervals $(0, z_{1,1})$, $(z_{1,1}, 1)$, $(1, z_{1,2})$, $(z_{1,2}, z_{1,3})$, $(z_{1,3}, \infty)$.

Continuing in this manner, we find that for $i = 3, 4, \ldots, N-1$, the $2i+1$ zeros of $Q_i(z)$, $z_{i,1}$, $z_{i,2}$, ..., $z_{i,2i+1}$, are real and distinct; the first $i$ of them lie in the consecutive intervals between points 0, $z_{i-1,1}$, $z_{i-1,2}$, ..., $z_{i-1,i-1}$, 1; the other $i+1$ are in the intervals between points 1, $z_{i-1,i}$, $z_{i-1,i+1}$, ..., $z_{i-1,2i-1}$), $\infty$. Moreover, the sign of $Q_{i+1}(z)$ at point $z_{i,s}$ is $(-1)^{i+s+1}$ for $s = 1, 2, \ldots, i$, and $(-1)^{s-i}$ for $s = i+1, i+2, \ldots, 2i+1$.

The final polynomial, $Q_N(z) = D(z)$, of degree $2N+1$, has a root in each of the consecutive intervals between points 0, $z_{N-1,1}$, $z_{N-1,2}$, ..., $z_{N-1,N-1}$; it is negative at $z_{N-1,N-1}$ and is 0 at $z = 1$. The positions of the other $N+1$ roots depend on the value of the derivative $D'(1)$. If it is positive, then for some sufficiently small $\varepsilon$, $D(1-\varepsilon) < 0$ and $D(1+\varepsilon) > 0$; in that case, the other $N+1$ roots are in the consecutive intervals between points $1+\varepsilon$, $z_{N-1,N}$, $z_{N-1,N+1}$, ..., $z_{N-1,2N-1}$, $\infty$. If, on the other hand, $D'(1) < 0$, then for some sufficiently small $\varepsilon$, $D(1-\varepsilon) > 0$ and $D(1+\varepsilon) < 0$; that would mean that there is an extra, $N$'th root in the interval (0,1) and only $N$ roots greater than 1.

According to (15), we have $D'(1) = \bar{D}(1)$. That quantity can be obtained in closed form by performing he following steps: add the first row of $\bar{D}(1)$ to the second, the resulting second row to the third, ..., the $N-1$st to the $N$th; add rows $2, 3, \ldots, N$ and subtract them from row $N+1$. This leads to the expression

$$\bar{D}(1) = \begin{vmatrix} N\mu_2 & -\mu_1 q & & & & & \\ 0 & (N-1)\mu_2 & -2\mu_1 q & & & & \\ & 0 & (N-2)\mu_2 & -3\mu_1 q & & & \\ & & & \ddots & \ddots & & \\ & & & & \mu_2 & -N\mu_1 q & \\ -\lambda & \mu_1 - \lambda & 2\mu_1 - \lambda & & & N\mu_1 - \lambda & \end{vmatrix} . \quad (30)$$

Expanding this determinant along the elements of the last row yields, with a little manipulation,

$$\bar{D}(1) = D'(1) = N!(\mu_1 q + \mu_2)^{N-1}[N\mu_1\mu_2 - \lambda(\mu_1 q + \mu_2)] \ . \quad (31)$$

The term in square brackets in the right-side of (31) is positive exactly when the stability condition (1) is satisfied. Therefore, under that condition, $D(z)$ has $N-1$ distinct real roots in the open interval (0,1), and no other roots in the interior of he unit disc. The Lemma is established and a normalizeable solution to the balance equations exists.