

# Modelling Restricted Processor Sharing

Feng Zhang and Lester Lipsky

Department of Computer Science and Engineering, 371 Fairfield Rd, Unit 2155  
University of Connecticut, Storrs, CT 06269-2155  
{fzhang,lester}@engr.uconn.edu

## Abstract

*In principle unrestricted processor sharing can be very useful when jobs with widely varying CPU requirements are competing for the same processor. Even if there are several processors available, processor sharing can be useful. However, in practice it must be implemented by round-robin, and there is an overhead cost (e.g., cache thrashing) to implementing this scheme. Furthermore, the overhead may depend on the number of jobs that are active, and can be significant. Therefore restricted processor sharing, which only allows a limited number of jobs to share the processors, may be a more appropriate strategy. In this paper we present a comprehensive analytic model to study the interplay among the number of parallel processors, the maximum degree of processor sharing, the overhead, and the job arrival rate. We examine how the CPU time distribution affects mean system time (or response time), under what conditions two slow processors are better than one double fast one, and when it pays to invoke restricted processor sharing.*

Keywords: Restricted processor sharing, Job switching overhead, Linear Algebraic approach to Queueing Theory (LAQT)

## 1 Introduction

In the last ten years or so, it has become clear that service times of many types of jobs (e.g., running times of optimization programs and durations of Internet processes) have high variance (i.e., large coefficient of variation  $C_v^2 = \sigma^2/\bar{x}^2$ , where  $\bar{x}$  is mean service time and  $\sigma$  is standard deviation) [5]. According to the Pollaczek-Khinchin formula,

$$\bar{T} = \frac{\bar{x}}{1-\rho} + \frac{\bar{x}\rho}{1-\rho} \frac{C_v^2 - 1}{2} \quad (1)$$

the mean system time  $\bar{T}$  for an open  $M/G/1$  queue using the first-come-first-served (FCFS) strategy is proportional to  $C_v^2$ . In the formula,  $\rho$  ( $0 < \rho < 1$ ), the utilization parameter, is  $\lambda\bar{x}$  with  $\lambda$  being the mean arrival rate. Since  $C_v^2$  is large for many applications (with  $C_v^2 > 100$  being not unusual),  $\bar{T}$  will become quite large even under moderate loads. On the other hand, if unrestricted *processor sharing* is employed,  $\bar{T}$  is the same as that for  $M/M/1$  (i.e., Eq. (1) with  $C_v^2 = 1$ ) [4]. Hence, unrestricted *processor sharing* improves performance greatly if  $C_v^2 \gg 1$ .

Unrestricted *processor sharing* assumes that each job gets  $1/k$  of the total computing power if there are  $k$  jobs in the total, and no overhead is incurred. However, in practice it must be implemented by round-robin, and there are two sources of overhead cost to implementing this scheme. One type of overhead is incurred by system activities for switching among the jobs, such as saving and restoring of registers, putting a preempted job in a queue, and selecting another job for execution. The other type is cache thrashing overhead, which may depend on the number of jobs that are active. For instance, in all modern computer processors, there is a small amount of high-speed cache between CPU and the main memory. If only one or two jobs are active, it is possible that their working sets can all be put in the cache simultaneously such that the cache thrashing

overhead is negligible. If several jobs are active, the cache may not be able to simultaneously hold all their working sets such that the cache thrashing overhead can be significant.

For the ease of discussion, we use the job switching overhead to broadly denote both types of overhead in round-robin. Let  $\Delta$  be the time slice and  $O_T(k)$  be a function for deciding the overhead given  $k$  active jobs. The overhead percentage is denoted by  $\varepsilon(k)$ , which equals to  $O_T(k)/\Delta$ . We need  $O_T(k) \ll \Delta \ll \bar{x}$  for the processor sharing model to be realistic for analyzing the performance of round-robin. It is generally no problem to ensure  $\Delta \ll \bar{x}$  for many applications. However, for some  $k$ ,  $O_T(k)$  may not be much smaller than  $\Delta$ . As such, processor sharing of many jobs should be avoided to keep  $O_T(k)$  and  $\varepsilon(k)$  small. This leads to *restricted processor sharing*, which only allows a limited number of jobs to share the processors. In the following, let  $PS(K)$  denote *restricted processor sharing* ( $K > P$ , where  $P$  is the number of parallel processors and  $K$  the maximum degree of processor sharing) and  $PS(\infty)$  denote unrestricted *processor sharing*.

In this paper, we study the performance of  $PS(K)$  in a uniprocessor environment ( $P = 1$ ) as well as in a multiprocessor environment ( $P > 1$ ). The following assumptions are made. First, job arrivals are Poisson (with job arrival rate  $\lambda$ ). Second, the CPU time requirements of jobs are drawn from the same distribution  $G$ , which can be non-exponential. However, the exact CPU time requirement of any job is not known until its completion. Third, each processor has its own cache, but all the processors share the same main memory. Finally, since each processor has its own cache, the job switching overhead would be considered only when the number of active jobs is more than the number of processors. Using the notation of queueing theory, such an environment can be represented by an open  $M/G/P$  queue. We present a flexible analytical model of  $PS(K)$  for any CPU time distribution  $G$  with finite  $C_v^2$  and apply it to study the interplay among  $P$ ,  $K$ ,  $\varepsilon(k)$  ( $k \in [P + 1, K]$ ), and  $\lambda$ . We examine how the CPU time distribution affects mean system time  $\bar{T}$ . It is demonstrated that  $PS(K)$  is a more appropriate strategy for applications with large  $C_v^2$ . As a case study, we consider the following scenario: suppose a user wants to buy a computer that can handle highly varying job demands effectively. The user only has enough budget to buy either a computer with two slow processors or one with a double fast processor. Since these two computers have the same computing capacity, does that mean they have the same performance? By applying our model, we show that the answer is no. Specifically, the computer with two slow processors will outperform that with a double fast processor under moderately heavy job loads.

In the following, the related work is summarized first. Next, our analytical model is presented. Section 4 describes the numerical results. Finally, conclusions are given.

## 2 Related Work

If the CPU time requirements of jobs are known, more efficient scheduling strategies can often be employed. For example, the well-known shortest-remaining-processing-time (*SRPT*) strategy was shown to outperform processor sharing for heavy-tailed job size distributions [3]. However, if the CPU time requirement of each job is not available beforehand, other strategies, such as processor sharing and foreground-background (favoring jobs with less attained times) [1], have to be used.

Many researchers studied  $PS(\infty)$  and its variants. Most of them focus on  $M/G/1$  queues (see, for example, [1, 3, 7, 8, 14]). A class of multi-level scheduling disciplines that extend processor sharing was presented in [8] and analytical solutions were given for the mean system time of jobs conditioned on their service requirements by restricting the service time distribution to be Erlangians. Such extensions favor the jobs with less service requirements, e.g., many interactive applications. Along this track, [1] compared the performance of several two-level processor sharing disciplines with that of  $PS(\infty)$  by considering general job service distributions with either decreasing or increasing hazard rate. The case of bulk arrivals was studied in [2] for the generalized hyper-exponential distributions. In addition, processor sharing was studied in Jackson networks [4], finite-capacity  $M/M/1$  [9],  $GI/G/1$  [12], and so on.

However,  $PS(\infty)$  is not realistic because of the job switching overhead in the case of multiple active jobs. A constant overhead per time slice was assumed in [6] for studying two time-sharing algorithms while a linear overhead pattern was studied in [10] for analyzing performance bottlenecks in a computer system.

Both of them focused on an exponential service distribution.

Compared to the past work, we take a more practical approach by studying the performance of  $PS(K)$  in  $M/G/P$  queues ( $P \geq 1$ ) with job service distributions having large  $C_v^2$  (i.e.,  $C_v^2 \gg 1$ ) and taking into account the job switching overhead. A detailed model of the overhead as a function of the number of active jobs is itself an interesting research. In this paper, we merely consider two types of overhead patterns: a constant overhead pattern and a two-level overhead pattern. In the following, an analytical model is presented to study the performance of  $PS(K)$  under non-negligible overhead.

### 3 Analytical Model

This model is essentially an application of the matrix-based solution methodology described in Linear Algebraic approach to Queueing Theory (*LAQT*) [11, 13]. It is flexible in that it can handle any probability distribution function with finite  $C_v^2$ , and can be easily applied to  $M/G/P$  ( $P \geq 1$ ) queues using  $PS(K)$  and *FCFS* scheduling strategies. The model can also be generalized to study more complicated systems [11]. We assume that readers have some knowledge of queueing theory, particularly continuous time Markov chains, which is helpful for understanding the model. However, those not familiar with queueing theory may safely skip the remaining section in the first pass and continue on the numerical result section.

Let  $S_1$  be a subsystem consisting of  $P$  identical processors (or servers). We assume that the CPU time distribution  $G$  is an *ME* (matrix-exponential) distribution with mean time  $\bar{x}$ , represented by  $m$  ( $\geq 1$ ) exponential phases. Conceptually, we say that each processor is made up of  $m$  phases. In this model, we keep track of which phases the active jobs are in. Since we do not distinguish individual jobs, it is sufficient to know the number of jobs in each phase. Hence, each state of  $S_1$  is denoted by an  $m$ -tuple  $\langle \alpha_1, \alpha_2, \dots, \alpha_m \rangle$ , where  $\alpha_l$  ( $l \in [1, m]$ ) is the number of jobs at phase  $l$ . The key information of describing a system, including the CPU time distribution, the mean service rate at each state, and the state transition probabilities, is represented by different kinds of matrices. In the following, the necessary notation is first described:

- $C$  is the maximal number of active jobs granted access to the system  $S_1$  at any time. If *FCFS* is used,  $C$  equals to  $P$ . If *restricted processor sharing* is employed, the maximum number of jobs allowed to share the processors,  $K$ , is assumed to be more than  $P$ . In this case,  $C$  equals to  $K$ .
- $\Xi_k$  ( $k \in [1, C]$ ) is the set of all states when there are  $k$  active jobs in the system  $S_1$ . A state  $i$  ( $= \langle \alpha_1, \alpha_2, \dots, \alpha_m \rangle$ ) is in  $\Xi_k$  if and only if  $\sum_{l=1}^m \alpha_l = k$ . The total number of states in  $\Xi_k$  is denoted by  $D(k)$ , which equals to  $\binom{k+m-1}{k}$ . For example,  $D(1) = |\Xi_1| = m$ ,  $D(2) = |\Xi_2| = \frac{m(m+1)}{2}$ . Let  $\Xi_0 = \{ \langle 0, 0, \dots, 0 \rangle \}$ , which means that the system is idle.
- $\mathbf{M}_k$  is the diagonal completion rate matrix where  $[\mathbf{M}_k]_{ii}$  is the service rate (or completion rate) at state  $i$  ( $i \in \Xi_k$ ) and the rest elements are zeros.
- $\mathbf{P}_k$  is the transition matrix where  $[\mathbf{P}_k]_{ij}$  ( $i, j \in \Xi_k$ ), is the probability that a job, upon completing at some phase of a processor, goes to another phase such that the system changes from state  $i$  ( $\in \Xi_k$ ) to state  $j$  ( $\in \Xi_k$ ).
- $\mathbf{Q}_k$  is the exit matrix where  $[\mathbf{Q}_k]_{ij}$  is the probability that upon leaving of a job, the system goes from state  $i$  ( $\in \Xi_k$ ) to state  $j$  ( $\in \Xi_{k-1}$ ). In the case of  $k = 1$ ,  $\mathbf{Q}_1$  is alternatively called the exit vector, denoted by  $\mathbf{q}$ .
- $\mathbf{R}_k$  is the entrance matrix where  $[\mathbf{R}_k]_{ij}$  is the probability that a job, upon entering the system, goes to some phase of a processor so that the system changes from state  $i$  ( $\in \Xi_{k-1}$ ) to state  $j$  ( $\in \Xi_k$ ). In the case of  $k = 1$ ,  $\mathbf{R}_1$  is alternatively called the entrance vector, denoted by  $\mathbf{p}$ .
- $\mathbf{B}_k$  is defined to be  $\mathbf{M}_k(\mathbf{I}_k - \mathbf{P}_k)$ , where  $\mathbf{I}_k$  is the  $D(k) \times D(k)$  identity matrix.  $\mathbf{V}_k$  denotes the inverse of  $\mathbf{B}_k$ , i.e.,  $\mathbf{V}_k = \mathbf{B}_k^{-1}$ .

- $\epsilon_k = (1, 1, \dots, 1)$  is a special row vector of dimension  $D(k)$ .

With the above notation, the next step is to construct the matrices based on the application parameters (e.g., the CPU time distribution), the system parameters (e.g., the number of processors in  $S_1$  and the job switching overhead pattern), and the queueing discipline. The key is to decide the effective service rate at each state, which is reflected in the completion rate matrix  $\mathbf{M}_k$  ( $k \in [1, C]$ ). Under the different queueing disciplines and overhead patterns, the adjustment of  $\mathbf{M}_k$  ( $k \in [1, C]$ ) is generally different. Let  $\mu_j$  ( $j \in [1, m]$ ) be the mean service rate of phase  $j$  of each processor. On one hand, if *FCFS* is used, the mean service rate at state  $i = \langle \alpha_1, \alpha_2, \dots, \alpha_m \rangle$  ( $\in \Xi_k$ , where  $k \in [1, P]$ ) is  $[\mathbf{M}_k]_{ii} = \sum_{v=1}^m \alpha_v \mu_v$ . Note that, if  $k = 1$ , only one of the  $P$  processors is busy and  $[\mathbf{M}_k]_{ii} = \mu_i$ . On the other hand, if *PS(K)* ( $K > P$ ) is employed, the computation is slightly complicated. While  $\mathbf{M}_k$  ( $k \in [1, P]$ ) is the same as that of *FCFS* (since negligible overhead is assumed with no more than  $P$  active jobs), the incurred overhead  $\varepsilon(k)$  has to be taken into account for  $k \in [P + 1, K]$ . Specifically, the effective service rate at state  $i$  is  $[\mathbf{M}_k]_{ii} = (1 - \varepsilon(k))C/k \sum_{v=1}^m \alpha_v \mu_v$ . After deciding  $\mathbf{M}_k$  ( $k \in [1, C]$ ),  $\mathbf{P}_k$ ,  $\mathbf{Q}_k$ , and  $\mathbf{R}_k$  as well as other matrices can be derived accordingly. For example, if  $S_1$  has two identical processors, the service time distribution is Erlangian-2 with  $\bar{x} = 2$ , *PS(3)* is used, and a constant overhead  $\varepsilon_0$  (i.e.,  $\varepsilon(k) = \varepsilon_0, k > 2$ ) is incurred, then we can construct the matrices as follows:

$$\begin{aligned} \Xi_1 &= \{ \langle 1, 0 \rangle, \langle 0, 1 \rangle \}, \mathbf{M}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{P}_1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \mathbf{R}_1 = \mathbf{p} = (1, 0), \mathbf{Q}_1 = \mathbf{q} = (0, 1)' \\ \Xi_2 &= \{ \langle 2, 0 \rangle, \langle 1, 1 \rangle, \langle 0, 2 \rangle \} \\ \mathbf{M}_2 &= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \mathbf{P}_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0.5 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{R}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \mathbf{Q}_2 = \begin{bmatrix} 0 & 0 \\ 0.5 & 0 \\ 0 & 1 \end{bmatrix} \\ \Xi_3 &= \{ \langle 3, 0 \rangle, \langle 2, 1 \rangle, \langle 1, 2 \rangle, \langle 0, 3 \rangle \} \\ \mathbf{M}_3 &= \begin{bmatrix} 2(1 - \varepsilon_0) & 0 & 0 & 0 \\ 0 & 2(1 - \varepsilon_0) & 0 & 0 \\ 0 & 0 & 2(1 - \varepsilon_0) & 0 \\ 0 & 0 & 0 & 2(1 - \varepsilon_0) \end{bmatrix}, \mathbf{P}_3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 2/3 & 0 \\ 0 & 0 & 0 & 1/3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{R}_3 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \mathbf{Q}_3 = \begin{bmatrix} 0 & 0 & 0 \\ 1/3 & 0 & 0 \\ 0 & 2/3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

After constructing the matrices, we can compute the mean system time (or response time) of jobs using the following formulas (refer to Chapter 6 of [11] for details):

$$\bar{T} = \frac{r_0}{\lambda} \left\{ \sum_{n=1}^{C-1} n \mathbf{x}_n \epsilon'_n + x_C \mathbf{U}_C^{-C+1} \left[ (\mathbf{I}_C - \mathbf{U}_C)^{-2} - \sum_{n=1}^{C-1} n \mathbf{U}_C^{n-1} \right] \epsilon'_C \right\} \quad (2)$$

$$\frac{1}{r_0} = 1 + \sum_{n=1}^{C-1} \mathbf{x}_n \epsilon'_n + \mathbf{x}_C (\mathbf{I}_C - \mathbf{U}_C)^{-1} \epsilon'_C \quad (3)$$

$$\mathbf{x}_1 = \mathbf{p} \mathbf{U}_1 \quad (4)$$

$$\mathbf{x}_n = \mathbf{x}_{n-1} \mathbf{R}_n \mathbf{U}_n, \quad n \in [2, C] \quad (5)$$

$$\mathbf{U}_C = \lambda [\lambda \mathbf{I}_C + \mathbf{B}_C - \mathbf{U}_C \mathbf{M}_C \mathbf{Q}_C \mathbf{R}_C]^{-1} \quad (6)$$

$$\mathbf{U}_n = \lambda [\lambda \mathbf{I}_n + \mathbf{B}_n - \mathbf{R}_{n+1} \mathbf{U}_{n+1} \mathbf{M}_{n+1} \mathbf{Q}_{n+1}]^{-1}, \quad n \in [1, C - 1] \quad (7)$$

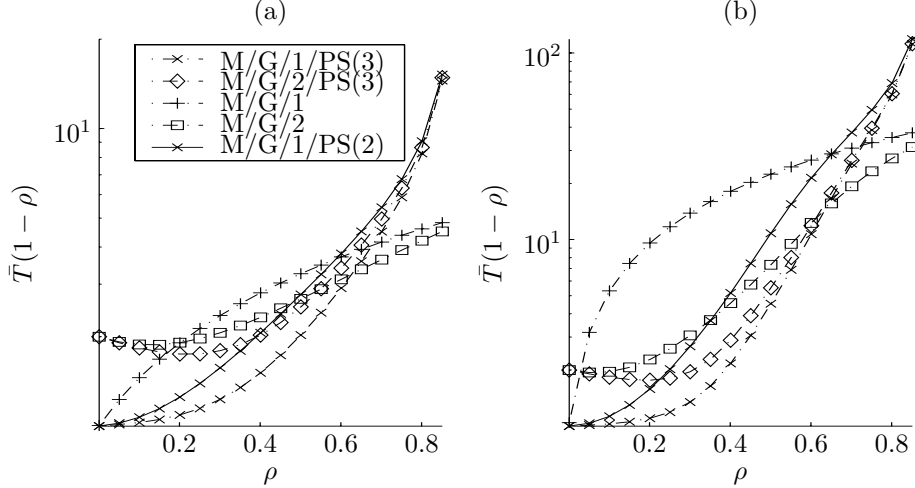


Figure 1: Under a fixed overhead,  $\varepsilon(k) = 0.1$ , the performance of  $PS(K)$  is improved by allowing more active jobs: (a) for hyper-exponential ( $C_v^2 = 10$ ); (b) for five-phase  $TPT$  ( $C_v^2 = 86.4$ ). Both graphs are plotted in semi-log scale on y-axis.

Equation (3) computes the probability of  $S_1$  being idle (i.e.,  $r_0$ ), where  $\mathbf{x}_n$  ( $n \in [1, C]$ ) are the auxiliary vectors. To compute the mean system time  $\bar{T}$ , first  $\mathbf{U}_C$  has to be calculated using Eq. (6). Since  $\mathbf{U}_C$  appears on both sides, an iterative approach is used to compute  $\mathbf{U}_C$  with the initial  $\mathbf{U}_C = \lambda \mathbf{V}_C$ . Then,  $\mathbf{U}_n$  ( $n \in [1, C - 1]$ ) can be computed backwards using Eq. (7). Finally,  $\mathbf{x}_n$  ( $n \in [1, C]$ ) can be computed using Eqs. (4,5) and  $\bar{T}$  can be derived thereby. In the case of an open  $M/G/1$  queue using  $FCFS$ , Eq. (2) reduces to Eq. (1).

## 4 Numerical Results

In this section, we apply the above model to show that  $PS(K)$  can improve the performance of an open  $M/G/P$  queue ( $P \geq 1$ ) with large  $C_v^2$ . As a case study, we compare the performance of a computer with two slow processors to that of a computer with a double fast processor. Two CPU time distributions with large  $C_v^2$  are considered: hyper-exponential and five-phase  $TPT$  [5]. The variance of a  $TPT$  distribution is finite but can be set to arbitrarily large by varying its parameters. On the limit, however, its variance can be infinite. For each distribution, we measure the performance of using  $PS(2)$  and  $PS(3)$  in a computer with a double fast processor as well as that of using  $PS(3)$  in a computer with two slow processors, denoted by  $M/G/1/PS(2)$ ,  $M/G/1/PS(3)$ , and  $M/G/2/PS(3)$ . As a comparison, the performance of  $M/G/1$  and  $M/G/2$  using  $FCFS$  is measured. To have a true reflection of their performance, two kinds of overhead patterns are considered and the mean system time  $\bar{T}$  is computed for each strategy under different load conditions by varying the job arrival rate  $\lambda$  ( $\rho = \lambda \bar{x}/P$ ). For simplicity, it is assumed that  $\bar{x} = P$ . We plot  $\bar{T} * (1 - \rho)$  for better illustration. It is shown that a computer with two slow processors outperforms a computer with one double fast processor in many cases. The results can provide guidance on selecting a suitable strategy according to the specific situation.

We first consider a constant overhead pattern, which is justified if the jobs are generally long and the time slice  $\Delta$  is sufficiently large. Figure 1.(a,b) shows that for hyper-exponential and five-phase  $TPT$ ,  $PS(3)$  outperforms  $PS(2)$  under a fixed overhead. It is expected that, by allowing more active jobs, the performance is even better. So on the limit,  $PS(\infty)$  should be chosen over  $PS(K)$ . From the figure, it can be seen that no one system configuration is always the best across different load conditions. Specifically, if the job load is heavy, a computer with two slow processors should be chosen and  $FCFS$  should be applied. Otherwise, a computer with a double fast processor using  $PS(\infty)$  is preferred.

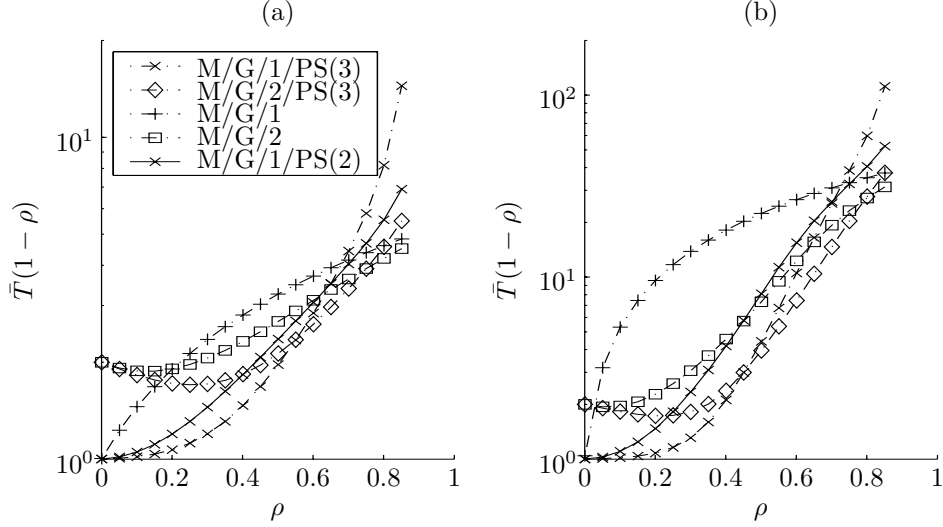


Figure 2: Performance comparison under a two-level overhead pattern: (a) hyper-exponential ( $C_v^2 = 10$ ); (b) five-phase  $TPT$  ( $C_v^2 = 86.4$ ). Both graphs are plotted in semi-log scale on y-axis. In the case of one double fast processor,  $\varepsilon_1(2) = 0.05$  and  $\varepsilon_1(3) = 0.10$ , while in the case of two slow processors,  $\varepsilon_2(2) = 0.0$  and  $\varepsilon_2(3) = 0.033$ .

A two-level overhead pattern seems appropriate if the overhead becomes significant beyond some point. Here, it is assumed that the cache of each processor can hold the working sets of 1.5 jobs, and 5% (or 10%) of the computing power is spent on the job switching overhead if half of the working set (or the whole working set) of a job has to be loaded during its execution. By this assumption, in the case of a computer with one double fast processor, there is a 5% (or 10%) overhead if two (or three) jobs are active. In the case of a computer with two slow processors, we assume that three active jobs get the equal share of two slow processors every three time slices: In the first time slice, jobs 1 and 2 are active; In the second time slice, jobs 1 and 3 are active; In the third slice, jobs 2 and 3 are active. As a result, a 3.33% overhead on average is estimated if three jobs are active.

Figure 2 compares various strategies under the two-level overhead pattern. From the figure, two crossover points can be identified. The first crossover point tells us that a computer with one double fast processor using  $PS(3)$  is preferred if the job load is light, and a computer with two slow processors using  $PS(3)$  should be chosen if the load is moderately heavy. The second crossover point tells us that a computer with two slow processors using  $FCFS$  should be chosen if the load is heavy. Another observation is that with the increase of  $C_v^2$ , a computer with two slow processors using  $PS(3)$  becomes more and more favored. For example, it is chosen when  $\rho$  is above 0.55 and below 0.75 for hyper-exponential, while the choice has been made when  $\rho$  is just above 0.45 for five-phase  $TPT$  and is not changed to  $FCFS$  until  $\rho$  is about 0.8. Furthermore, although  $PS(3)$  incurs more overhead than  $PS(2)$ , the performance of a computer with one double fast processor using  $PS(3)$  outperforms that of using  $PS(2)$  if the load is not heavy. This is because the system always has idle time and the overhead actually reduces the idle time without decreasing effective computing power much.

## 5 Conclusions

In this paper, a comprehensive analytic model is presented to study the performance of *restricted processor sharing* ( $PS(K)$ ) in  $M/G/P$  ( $P \geq 1$ ) queues with the focus on the interplay among  $P$ ,  $K$ , overhead,  $\lambda$  (job arrival rate), and variation of job times ( $C_v^2$ ) to see how they affect mean system time. It is demonstrated that  $PS(K)$  is a more practical strategy than  $PS(\infty)$  because it takes into account the job switching overhead,

and is effective in handling highly varying jobs demands (i.e.,  $C_v^2 \gg 1$ ). In the case study, we show that, by using  $PS(K)$ , a computer with two slow processors can outperform one with a double fast processor when the arrival rate is moderately heavy and the overhead increases with the number of active jobs. This conclusion can be generalized to the case of a multiprocessor computer system. Specifically, a multiprocessor system is often preferred to a uniprocessor system for applications with large  $C_v^2$  by assuming that they have the same computing power.

In the view that a computer system consists of not only processors, but also peripherals such as channels and disks, our ongoing work is to study the performance of  $PS(K)$  in a more realistic system. While it is not considered here, main memory thrashing could happen as well with the increase of the number of active jobs. We would like to address this issue in the future work. Furthermore, *restricted processor sharing* must be implemented by restricted round-robin in practice, which could be sensitive to the time slice value. Hence, another future work is to incorporate the influence of time slice in our model.

## References

- [1] S. Aalto, U. Ayesta, and E. Nyberg-Oksanen. Two-level processor-sharing scheduling disciplines: mean delay analysis. *SIGMETRICS Perform. Eval. Rev.*, 32(1):97–105, 2004.
- [2] N. Bansal. Analysis of the m/g/1 processor-sharing queue with bulk arrivals. *Operations Research Letters*, 31(5):401–405, 2003.
- [3] N. Bansal and M. Harchol-Balter. Analysis of srpt scheduling: Investigating unfairness. In *Proc. ACM Sigmetrics'01*, 2001.
- [4] 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. ACM*, 22(2):248–260, 1975.
- [5] M. Greiner, M. Jobmann, and L. Lipsky. The importance of power-tail distributions for modeling queueing systems. *Operations Research*, 47(2):313–326, 1999.
- [6] H. C. Heacox, Jr. and P. W. Purdom, Jr. Analysis of two time-sharing queueing models. *J. ACM*, 19(1):70–91, 1972.
- [7] M. Y. Kitaev. The m/g/1 processor-sharing model: transient behavior. *Queueing Systems*, 14:239–274, 1993.
- [8] L. Kleinrock and R. R. Muntz. Processor sharing queueing models of mixed scheduling disciplines for time shared system. *J. ACM*, 19(3):464–482, 1972.
- [9] C. Knessl. On the sojourn time distribution in a finite capacity processor shared queue. *J. ACM*, 40(5):1238–1301, 1993.
- [10] L. Lipsky. Use of queueing theory for modeling a computer system. In *Proc. Share XLII*, 1974.
- [11] L. Lipsky. *QUEUEING THEORY: A Linear Algebraic Approach*. McMillan, 1992.
- [12] M. Mandjes and A. Zwart. Large deviations of sojourn times in processor sharing queues. Technical report, CWI, Netherlands, PNA-E0410, 2004.
- [13] M. F. Neuts. *Matrix-geometric Solutions in Stochastic Models*. Johns Hopkins University Press, Baltimore, MD, 1981.
- [14] A. Ward and W. Whitt. Predicting response times in processor-sharing queues. In *Proceedings of the Fields Institute Conference on Communication Network*, 2000.