Analysis of Cycle Stealing with Switching Cost

Takayuki Osogami[1] Mor Harchol-Balter[2]
Alan Scheller-Wolf[3]

October 2002

CMU-CS-02-192

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We consider the scenario of two independent processors, each serving its own workload, where one of the processors (known as the "donor") can help the other processor (known as the "beneficiary") with its jobs, during times when the donor processor is idle. That is the beneficiary processor "steals idle cycles" from the donor processor. We assume that both donor jobs and beneficiary jobs may have generally-distributed service requirements. We assume that there is a switching cost required for the donor processor to start working on the beneficiary jobs, as well as a switching cost required for the donor processor to return to working on its own jobs. We also allow for threshold constraints, whereby the donor processor only initiates helping the beneficiary if both the donor is idle and the number of jobs at the beneficiary exceeds a certain threshold.

We analyze the mean response time for the donor and beneficiary processors. Our analysis is approximate, but can be made as close to exact as desired. Analysis is validated via simulation. Results of the analysis illuminate several interesting principles with respect to the general benefits of cycle stealing and the design of cycle stealing policies.

[1]Carnegie Mellon University, CSD. Email: osogami@cs.cmu.edu

[2]Carnegie Mellon University, Computer Science Department. Email: harchol@cs.cmu.edu.

This work was supported by NSF Career Grant CCR-0133077, by NSF ITR Grant 99-167 ANI-0081396 and by Spinnaker Networks via Pittsburgh Digital Greenhouse Grant 01-1.

[3]Carnegie Mellon University, GSIA. Email: awolf@andrew.cmu.edu

Keywords: cycle stealing, switching cost, queueing theory, task assignment, server farm, distributed system, unfairness, starvation, load sharing, supercomputing, Matrix-Analytic.

1 Introduction

Motivation

Since the invention of networks of workstations, systems designers have touted the benefits of allowing a user to take advantage of machines other than her own, at times when those machines are idle. This notion is often referred to as *cycle stealing*. Cycle stealing allows a user, Betty, with multiple jobs to offload one of her jobs to Dan's machine if Dan's machine is idle, giving Betty two machines to process her jobs. When Dan's workload resumes, Betty must return to using only her own machine. We refer to Betty as the *beneficiary*, to her machine as the beneficiary machine/server, and to her jobs as beneficiary jobs. Likewise, we refer to Dan as the *donor*, to his machine as the donor machine/server, and to his jobs as donor jobs.

Although cycle stealing provides obvious benefits to the beneficiary, these benefits come at some cost to the donor. For example, the beneficiary's job may have to be checkpointed and the donor's working set memory reloaded before the donor can resume, delaying the resumption of processing of donor jobs. In our model we refer to these additional costs associated with cycle stealing as *switching costs*.

A primary goal of this paper is to understand what is the benefit of cycle stealing for the beneficiary and what is the penalty to the donor, as a function of switching costs. A second goal is to derive parameter settings for cycle stealing. In particular, given non-zero switching costs, cycle stealing may only pay if the beneficiary's queue is "sufficiently" long. We seek to understand the optimal threshold on the beneficiary queue. More broadly, we seek general insights into which problem parameters have the most significant impact on the effectiveness of cycle stealing.

The analytical model

We assume two independent M/GI/1/FCFS queues, the *beneficiary* queue and the *donor* queue. Jobs arrive at rate λ_B (respectively, λ_D) at the beneficiary (respectively, donor) queue and have service requirement represented by the random variable X_B drawn from distribution G_B (respectively, X_D drawn from distribution G_D). The load made up by beneficiary (respectively, donor) jobs is denoted by ρ_B (respectively, ρ_D) where $\rho_B = \lambda_B \cdot E[X_B]$ and $\rho_D = \lambda_D \cdot E[X_D]$. If the donor server is idle, and if the number of jobs at the beneficiary queue is at least N_B^{th}, the donor transitions into the switching state, for a random amount of time, K_{sw}. After K_{sw} time, the donor server is available to work on the beneficiary queue and the beneficiary queue becomes an M/G_B/2 queue. If a donor job arrives during K_{sw}, or during the time the donor is helping the beneficiary, the donor transitions into a switching back state, for a random amount of time, K_{ba}. After the completion of the switch back, the donor server resumes working on its own jobs. We assume the first four moments of the service times are finite, and queues are stable. No work is accomplished by the donor server while switching.

A few details are in order. First, in the above model, the donor processor continues to cooperate with the beneficiary even if there is no beneficiary work left for it to do — the donor processor only switches back when a donor job arrives. (We also analyzed the case where the donor processor switches back when it is not needed, see Appendix A. We found that this has almost no effect on performance, even under high switching costs). Second, we assume that if the donor processor is working on a beneficiary job and a donor job arrives, that beneficiary job is returned to the beneficiary queue and will be resumed by the beneficiary processor as soon as that processor is available. The work done on the job is not lost, i.e., the job is checkpointed.[1] Finally, our model can be generalized to the case where there is a threshold, N_D^{th}, on the number of donor jobs as well — i.e., the donor processor only returns to the donor queue when the number of jobs at the donor queue is at least N_D^{th}. Throughout this paper, aside from the stability section (Section 4), we assume $N_D^{th} = 1$ for simplicity, and focus on the effect of N_B^{th}. In Appendix B, we extend the analysis to general N_D^{th}. Our performance metric throughout is mean response time for each class of jobs.

Difficulty of analysis and Previous work

Consider the simplest instance of our problem — where the service requirements of all jobs are each drawn from exponential distributions and the switching costs and threshold are zero. Even for this simplest instance the continuous-time Markov chain, while easy to describe, is mathematically intractable. This is due to the fact that the stochastic process having state *(number beneficiary jobs, number donor jobs)* grows infinitely in two dimensions and contains no structure that can be exploited to obtain an exact solution. While solution by truncation of the Markov chain is possible, the errors that are introduced by ignoring portions of the state space (infinite in two dimensions) can be quite significant, especially at higher traffic intensities.[2] Thus truncation is neither sufficiently accurate nor robust for our purposes.

To our knowledge, there has been no previous analytical work on the problem of cycle stealing with setup costs. Below we describe prior work on the "coupled processor model". In this model two processors each serve their own class of job, and if either is idle it may help the other, increasing the rate of this other processor. This help incurs no switching cost and has an effect if even a single job is present. These models inherently assume a preemptive resume discipline: when a processor returns to

[1] It is easy to generalize our analysis to the case where the beneficiary requires a "switching time" before it can resume a job that the donor started. We have chosen to ignore this generalization for purpose of clarity. It is also trivial to extend our results to the case where all work on the job in progress is lost if a donor job arrives, provided that we assume that the job is restarted with a new service time – which we feel is unrealistic. It is also possible to extend our results to the case where the donor must complete the beneficiary job in progress before it switches back. This is the topic of current research, see Section 7.

[2] For $\rho_B = 1.2$ and $\rho_D = 0.7$, truncation leads to $> 15\%$ error, even with 64^2 states, and takes > 10 minutes to compute. As ρ_B nears 1.3, the error increases indefinitely. Under jobs sizes more variable than exponential, the error increases.

its own queue, none of its work is lost. All works we mention below consider Poisson arrivals.

Early work on the coupled processor model was by Fayolle and Iasnogorodski [5] and Konheim, Meilijson and Melkman [6]. Both papers assume exponential service times, deriving expressions for the stationary distribution of the number of jobs of each type. Fayolle and Iasnogorodski use complex algebra, eventually solving either a Dirichlet problem or a homogeneous Riemann-Hilbert problem for a circle, depending on the accelerated rates of the servers. Konheim et. al. assume that the accelerated rate is twice the original rate, which yields simpler expressions (still in the form of complex integrals). No indication is given in either work as to how the evaluation of these expressions. In addition, because the processors work in concert, rather than on different jobs, these systems will gain no multi-server benefit when serving highly variable jobs; short jobs may get stuck waiting behind long jobs in a single queue.

The above work was extended by Cohen and Boxma [4] to the case of general service times. They consider stationary *workload*, which they formulate as a Wiener-Hopf boundary problem. This leads to complex expressions involving either integrals or infinite sums; if the queues are symmetric simpler expressions for mean total workload are found. They again have the two processors working in concert, without a switching cost, providing complex analytical expressions, rather than numerical values.

In more recent work, Borst, Boxma and van Uitert [3] apply a transform method to the expressions in [4], yielding asymptotic relations between the workloads and the service requirement distributions. This leads to the insight that if a processor has a load less than one, it is "insulated" from the heavy-tail of the other, as long periods without cooperation will not lead to large backlogs. This is not the case if the load is greater than one, as the queue now must rely on help to be stable. Borst, Boxma and Jelenkovic [2] consider a very similar question under generalized processor sharing. Using probabilistic bounds, they show that different service rates can either insulate the performance of different classes from the others or not, again depending on whether the minimal rate is larger than the load. Both of these papers are concerned with the asymptotic behavior of workload, whereas our work isolates mean performance. Our work is thus complementary to these results.

Our approach

This paper presents the first analysis of cycle stealing under general service requirements with switching costs and a switching threshold. Recall that the difficulty in analyzing cycle stealing is that the corresponding stochastic process defines a Markov chain which grows infinitely in two dimensions (2D), making it intractable. The key idea in our approach is to find a way to transform this 2D chain into some 1D infinite Markov chain which can be analyzed. The questions in applying such a transformation are (i) what should the 1D Markov chain track, and (ii) how can all the relevant information from the 2D chain be captured in the 1D chain. Our 1D chain tracks the number of beneficiary jobs. For the donor jobs, our state space contains only binary knowledge: zero jobs or ≥ 1 jobs. Nevertheless we are able to capture detailed information on the number of donor jobs by using special *transitions* in our Markov chain, where these transitions represent the lengths of an assortment of busy periods. The difficulty lies in specifying the right busy periods, some of which transcend the definition of the analytical model.

Once the 1D Markov chain is specified, the hard work is finished, since this chain can be solved efficiently using known numerical (Matrix analytic) techniques. While a closed-form solution may be preferable, our chain is compact enough, and Matrix analytic methods powerful enough, that only a couple seconds are required to generate any of the results plots in this paper. Furthermore, our method very easily generalizes to more complex problem formulations *e.g.*, multiple donors (Section 7).

Our analysis is approximate, but can be made as close to exact as desired. The only approximation lies in representing the length of the busy periods by a Coxian distribution fit to a finite number of the busy period moments. In this paper, we use a 2-stage Coxian to capture the first 3 moments of the busy periods, and verify that this is sufficient via simulation. However, our method naturally extends to matching more moments.

Our analysis assumes a Poisson arrival process for both classes of jobs. The service requirements of the each class are assumed to be drawn i.i.d. from general distributions (which we approximate by a Coxians). The arrival process can easily be generalized to a MAP – Markovian Arrival Process [1].

Summary of results

Our analysis yields many interesting results concerning cycle stealing, detailed in Section 6. While cycle stealing obviously benefits the beneficiaries and hurts the donors, we find that when $\rho_B > 1$, cycle stealing is profitable overall even under significant switching costs as it may ensure stability of the beneficiary queue. For $\rho_B < 1$, we define load-regions under which cycle stealing pays. We find that in general the switching cost is only prohibitive when it is large compared with $E[X_D]$. Under zero switching cost, cycle stealing always pays.

Two counterintuitive results are that when $\rho_B < 1$, the performance of the beneficiaries is surprisingly insensitive to the switching cost, and also insensitive to the variability of the donor job size distribution. Even when the variability of the donor job sizes is very high, and donor help is very bursty, the beneficiaries don't seem to suffer.

We also study the effect of N_B^{th}, and present several rules of thumb for choosing the optimal setting of N_B^{th} as a function of the server loads, the switching cost, and the mean job sizes.

Outline

In Section 2 we present our analysis for the case of zero switching cost, generalizing to non-zero cost in Section 3. In Section 4 we provide stability conditions for the donor and beneficiary hosts. In Section 5 we validate our analysis by considering limiting analytical cases and via simulation. In Section 6 we present results for mean response times of donor and beneficiary jobs under various loads, job size distributions, switching costs, and thresholds. Section 7 describes extensions to the model.

2 Analysis without switching cost

In this section we analyze the simpler case of zero switching cost. Figure 1 shows our Markov chain for the case where $N_B^{th} = 3$, i.e. the donor server switches to help beneficiary jobs when there are zero donor jobs and there are at least three beneficiary jobs. Figure 1(a) shows a simplified form of our chain where job sizes and busy periods are assumed to be exponentially-distributed. Figure 1(b) and (c) show the case of generally-distributed (Coxian) job sizes and busy periods. The actual chain that we evaluate in the paper is a superposition of the chains in Figure 1(b) and (c), shown in Appendix C.

The first two components of each state denote the number of beneficiary jobs and the number of donor jobs respectively. The states of the Markov chain have been grouped into three rows, labeled (i) *cooperating row*: indicates that the donor processor is cooperating with the beneficiary; (ii) *independent row, with* ≥ 1 *donor job*: indicates that the donor and beneficiary processors are each working independently on their own queues and there is at least 1 donor job present; (iii) *independent row, with* 0 *donor jobs*: indicates that the donor and beneficiary processors are each working independently on their own queues and there are zero donor jobs present.

Observe that while the states track the precise number of beneficiary jobs, they keep only a binary record (zero or ≥ 1) of the donor jobs. The key idea is that to determine beneficiary performance we can avoid tracking the number of donor jobs because we only need to know when the donor queue is idle. Thus we use transitions (labeled B_D) to represent the length of a busy period of donor jobs.

The logic behind the Markov chain in Figure 1(a) is as follows: If we are in the row where the processors are working independently and the number of donor jobs is zero, the left-right transitions allow us to track the number of beneficiary jobs. If a donor job arrives we transition to the row where processors are working independently and the number of donor jobs is at least one. The time spent in this row is the length of a donor busy period. If at the end of the busy period the number of beneficiary jobs is below N_B^{th}, then we return to the row where processors are working independently and the number of donor jobs is zero. If however at the end of the busy period the number of beneficiary jobs is at least N_B^{th}, then we move to the cooperating row, where we stay until there is a donor arrival.

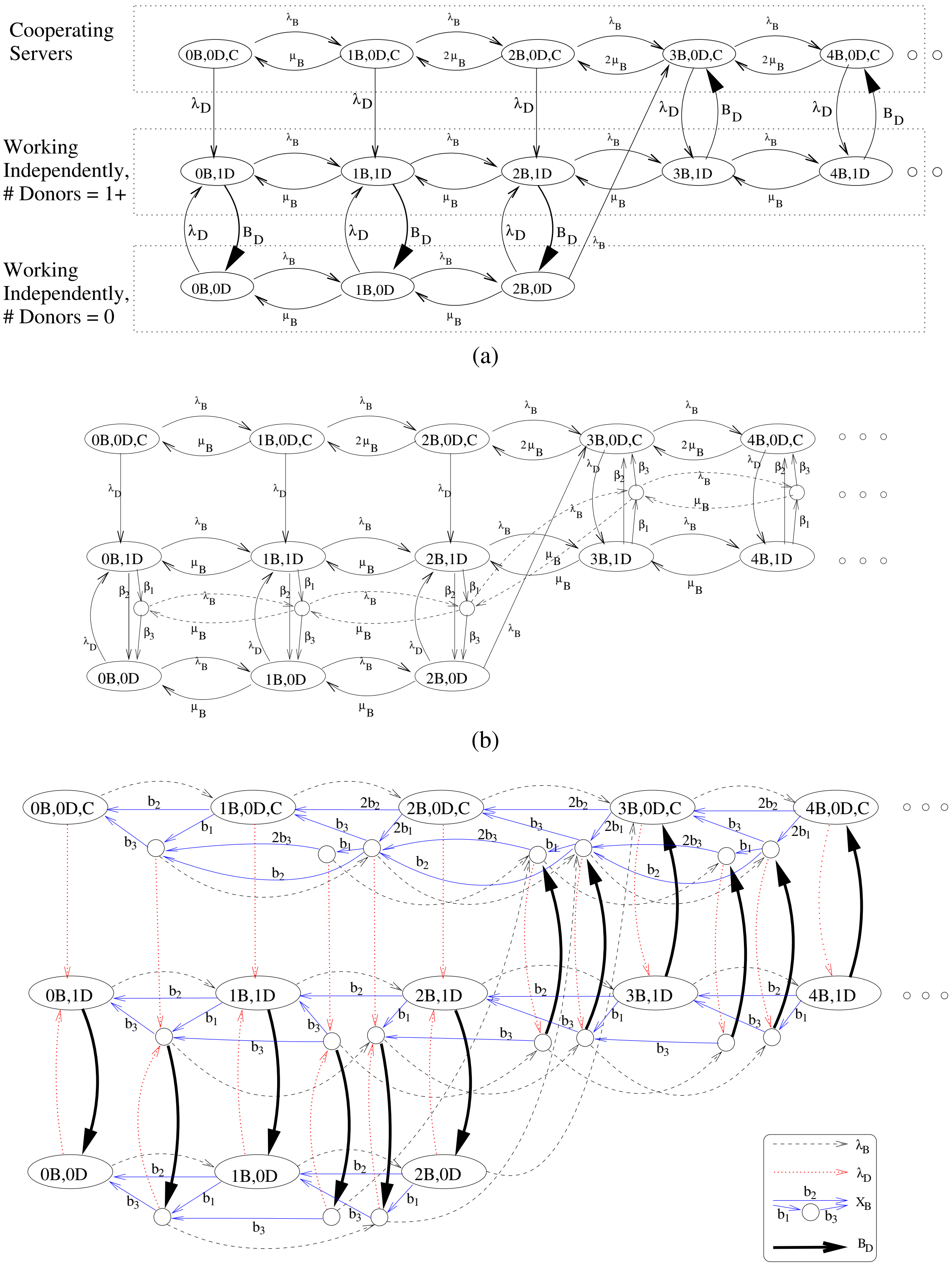


Figure 1: *Markov chain for cycle stealing without switching cost. (a) Where X_B is exponential and B_D is drawn using a single (exponential) transition. (b) Where X_B is exponential, and B_D is represented by a 2-stage Coxian. (c) Where X_B is represented by a 2-stage Coxian and B_D is drawn using a single (exponential) transition.*

Figure 1(b) and (c) show the same Markov chain as 1(a), where now X_B and B_D are represented by 2-stage Coxian distributions. Observe that when the donor and beneficiary are cooperating, the state space now must maintain a state for two partially-completed beneficiary jobs. If a donor job arrives while we're in the cooperating row, the partially-completed job that the donor was working on will be moved to the head of the beneficiary queue (we currently assume zero cost for the transfer, but this is easy to generalize to non-zero cost).

In order to specify the Markov chain, we need to compute the first three moments of B_D and then find a 2-stage Coxian which matches these. The Laplace transform of B_D is:

$$\widetilde{B_D}(s) = \widetilde{X_D}(s + \lambda_D - \lambda_D \widetilde{B_D}(s))$$

The moments of B_D are obtained from the transform by taking derivatives. In [9] we derive necessary and sufficient condition for matching the first three moments of a distribution using a 2-stage Coxian.

2.1 Computing response times

The mean response time for donor jobs is trivial to compute – it is simply the response time of the M/G_D/1 queue, since the beneficiary jobs are preemptive and switching cost is zero.

The mean number of beneficiary jobs is easy to compute from the chain in Figure 1 using the Matrix-Analytic method, described in [8] and [7]. This is a simple, compact and efficient method for solving QBD (quasi-birth-death) Markov chains which are infinite in only one dimension, where the chain repeats itself after some point, as does Figure 1. We then apply Little's Law to get their mean response time. Every plot in this paper which uses Matrix-Analytic analysis (to solve multiple instances with different parameter values) was produced within a couple of seconds using the Matlab 6 environment.

3 Analysis with switching cost

In this section, we analyze cycle stealing with switching costs in both directions. We assume that (i) the donor only makes the switch if the donor queue is idle and the number of jobs at the beneficiary queue is at least N_B^{th}, and (ii) the donor stays at the beneficiary queue until there is a donor arrival.

Let K_{sw} denote the time required for the donor to switch to working on the beneficiary queue, and K_{ba} the time to switch back to the donor queue. Figure 2 shows the Markov chain for cycle stealing with switching cost where $N_B^{th} = 3$. For simplicity, we have drawn X_B and K_{sw} as being exponentially-distributed — these are easy to replace with Coxian distributions.

The first two rows of the Markov chain in Figure 2 represent the case where the donor and beneficiary servers are working independently, where row 2 indicates zero donor jobs and row 1 indicates at least

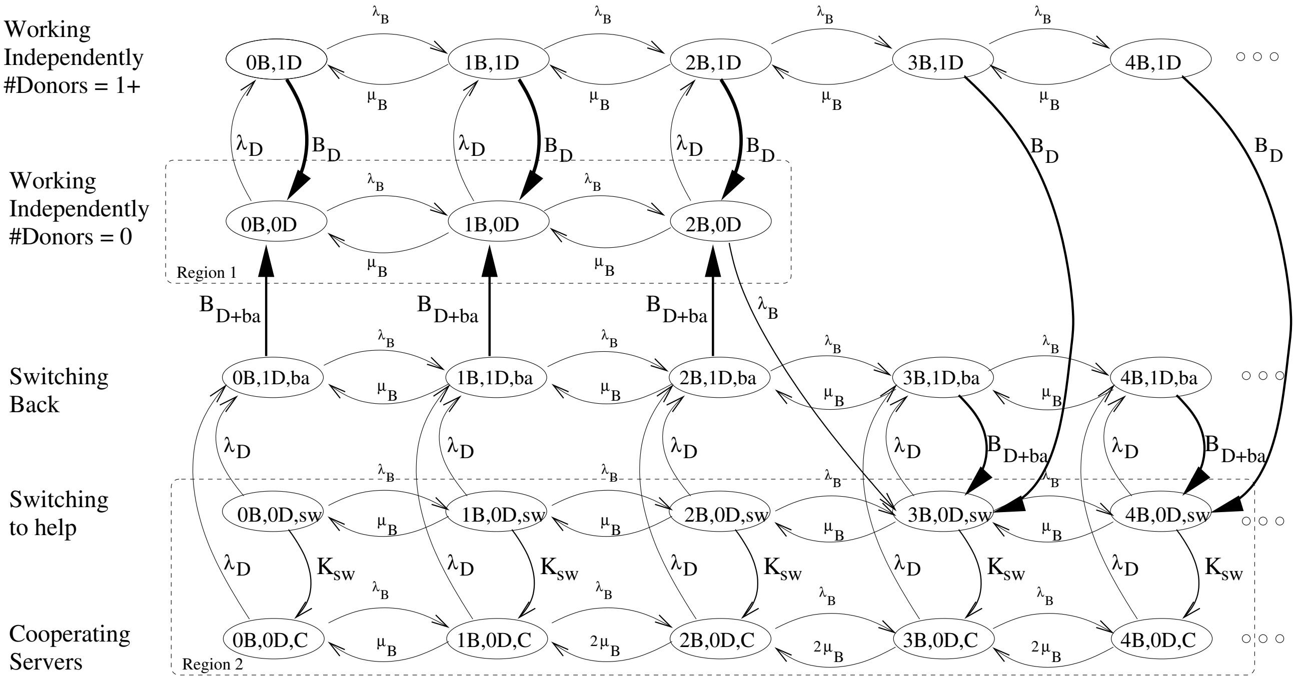


Figure 2: *Markov chain for cycle stealing with switching cost, X_B and K_{sw} exponential, $N_B^{th} = 3$.*

one donor job. A transition from row 2 to row 1 starts a donor job busy period, the length of which is represented by B_D. When this busy period completes, if there are $< N_B^{th}$ beneficiary jobs in the system, the Markov chain simply transitions back to row 2. However if there are $\geq N_B^{th}$ beneficiary jobs, the Markov chain transitions to row 4 – the row for switching to help. Observe that the Markov chain can also go from row 2, the zero donor jobs row, directly to row 4, the switching to help row, as soon as there are N_B^{th} beneficiary jobs. After K_{sw} time, if no donor job arrived, the Markov chain transitions to row 5, the cooperating row. As soon as a donor job arrives, either during K_{sw} or during the cooperating phase, the Markov chain immediately transitions into row 3 – the switching back row. At the point of entering row 3, there is a single donor job, which just arrived. The time to return to the case of zero donor jobs is thus a busy period started by the sum of X_D and K_{ba}. We denote this busy period by B_{D+ba}.

In our Markov chain, we have drawn the busy period transitions with a single bold transition, although in evaluating the chain we will replace this bold transition with a 2-stage Coxian as usual.[3] Computing the first three moments of a busy period of donor jobs, started by a job of size K_{ba}, the switch back time, is straightforward by taking derivatives of the Laplace transform:

$$\widetilde{B_{ba}}(s) \quad = \quad \widetilde{K_{ba}}(s + \lambda_D - \lambda_D \widetilde{B_D}(s))$$

[3] For some busy periods such as B_{D+ba}, there does not exist a 2-stage Coxian which matches the first 3 moments, see [9]. In such cases we divide B_{D+ba} into two busy periods: B_D, the busy period started by a donor job, and B_{ba}, the busy period started by switching back. We then may use two 2-stage Coxians to represent each busy period, since $B_{D+ba} = B_D + B_{ba}$.

Calculation of response times

The mean response time for beneficiary jobs is easy to compute, since the chain keeps track of the number of beneficiary jobs and can be analyzed via Matrix Analytic methods. The donor jobs see an M/GI/1 queue where, at times, the first job in a busy period must wait for a time to switch back, K_{ba}. Thus the response time for donor jobs is the response time under an M/GI/1 queue with setup time S:

$$S = \begin{cases} 0 & \text{with probability } a_1 = \frac{\Pr\{\text{Region 1}\}}{\Pr\{\text{Region 1 or 2}\}}, \\ \\ K_{ba} & \text{with probability } a_2 = \frac{\Pr\{\text{Region 2}\}}{\Pr\{\text{Region 1 or 2}\}}. \end{cases}$$

We consider only regions 1 and 2, as S is defined by what the first job to start a busy period sees. The expected waiting time for an M/GI/1 queue with only donor jobs and setup time S is known [10]:

$$E[W]^{M/GI/1/SetupS} = \frac{2E[S] + \lambda_D E[S^2]}{2(1 + \lambda_D E[S])} + \frac{\lambda_D E[X_D^2]}{2(1 - \rho_D)}.$$

We thus have:

$$E[\text{Response time for donor}] = E[X_D] + E[W]^{M/GI/1/SetupS}.$$

4 Stability

In this section we derive stability conditions on ρ_D and ρ_B for cycle stealing with switching costs and thresholds N_B^{th} and N_D^{th}.

Theorem 1 *The stability condition (necessary and sufficient) for donor jobs is* $\rho_D < 1$.

Proof: We first prove sufficiency. Assume $\rho_D < 1$. Let B_D denote the length of a busy period at the donor queue. We first consider the case $N_B^{th} = 0$. A busy period at the donor queue is started by a switching cost K_{ba} and N_D^{th} donor jobs. As $\rho_D < 1$, the mean length of a busy period is

$$E[B_D] = \frac{N_D^{th} E[X_D] + E[K_{ba}]}{1 - \rho_D} < \infty.$$

In this case B_D clearly has a proper distribution and thus the queue is positive regenerative, hence stable.

Next we consider the case $N_B^{th} > 0$. In this case $E[B_D]$ is smaller than in the case $N_B^{th} = 0$ because there will be donor busy periods in which the donor hasn't left the donor queue, implying (i) there is no switching back cost, and (ii) the busy period is started by only one donor job.

Necessity follows immediately from the fact that the donor queue is unstable for all $\rho_D \geq 1$. ■

Before we derive the stability condition on ρ_B, we prove a lemma allowing us to assume $N_B^{th} = 0$.

Lemma 1.1 *If the beneficiary queue is stable at $N_B^{th} = 0$, then it is stable at $N_B^{th} = n$, $\forall\, 0 \leq n < \infty$.*

Proof: Let $L_B^{(n)}(t)$ denote the number of beneficiary jobs at time t given $N_B^{th} = n \geq 1$. Consider a new process $\widehat{L}_B^{(n)}(t)$ in which no service is given by either server to a beneficiary job if there are $\leq n$ jobs present at the beneficiary queue. Note that $\widehat{L}_B^{(n)}(t)$ stochastically dominates $L_B^{(n)}(t)$. Along any sample path, $\widehat{L}_B^{(n)}(t)$ will be equal to n + $L_B^{(0)}(t)$. Therefore, if $L_B^{(0)}(t)$ is proper, so too is $\widehat{L}_B^{(n)}(t)$. ■

We now prove the stability condition on ρ_B for exponential K_{sw}.[4]

Theorem 2 *For exponential K_{sw}, the stability condition for the beneficiaries with donor threshold N_D^{th} is:*

$$\rho_B < 1 + \frac{1 - \min(\rho_D, 1)}{N_D^{th} + \lambda_D E[K_{ba}]} \left[N_D^{th} - \lambda_D E[K_{sw}] \left\{ 1 - \left(\frac{\lambda_D E[K_{sw}]}{1 + \lambda_D E[K_{sw}]} \right)^{N_D^{th}} \right\} \right].$$

Proof: By Lemma 1.1, we are allowed to assume $N_B^{th} = 0$. Let F denote the time average fraction of time that the donor helps the beneficiary. Then the strong law of large numbers can be used to show that a necessary and sufficient condition for stability of the beneficiary jobs is

$$\rho_B < 1 + F.$$

If $\rho_D \geq 1$, $F = 0$. Thus we assume $\rho_D < 1$ and derive F using renewal reward theory. Let's consider a renewal to occur every time the donor becomes idle. Recall $N_B^{th} = 0$. By renewal-reward theory:

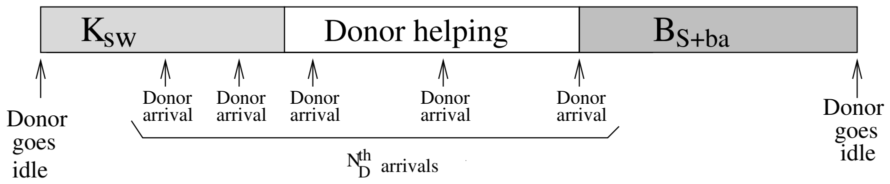
$$F = \text{Fraction of time donor helps beneficiary} = \frac{E[R]}{E[L]}$$

where R denotes the help (reward) during the renewal cycle, and L denotes the length of the renewal cycle. Observe that there may be any number of donor arrivals ranging from 0 to N_D^{th} during K_{sw} and we switch back only after the N_D^{th} arrival.

Let S denote the sum of the service times of N_D^{th} donor jobs. Then, as $\rho_D < 1$,

$$E[L] = N_D^{th} \cdot E[I_D] + E[B_{S+ba}] = N_D^{th} \cdot \frac{1}{\lambda_D} + \frac{N_D^{th} E[X_D] + E[K_{ba}]}{1 - \rho_D}$$

[4] For non-exponential K_{sw}, conditions can be derived in terms of expectations and probabilities.



where I_D is the interarrival time for donor jobs.

To compute $E[R]$ we condition on the number of donor arrivals during K_{sw}. If there are i arrivals, then the expected time spent helping is the time until there are $(N_D^{th} - i)$ more donor arrivals, $(N_D^{th} - i)I_D$.

Let p_i denote the probability that there are i donor arrivals during K_{sw}. As K_{sw} is exponentially-distributed,

$$p_i = \left(\frac{\lambda_D}{\lambda_D + \mu_{sw}}\right)^i \frac{\mu_{sw}}{\lambda_D + \mu_{sw}}.$$

Therefore,

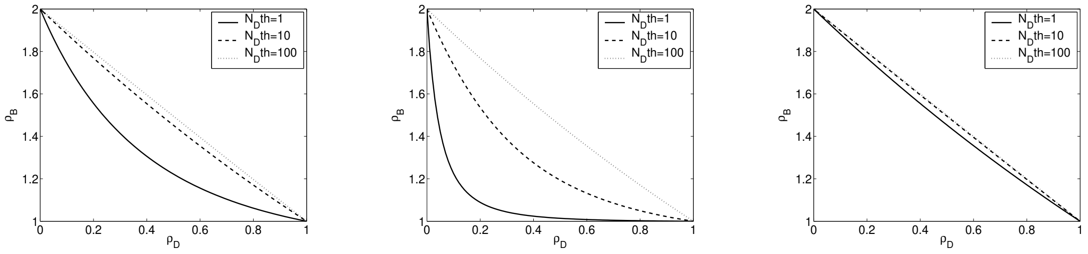
$$\begin{aligned} E[R] &= \sum_{i=0}^{N_D^{th}-1} (N_D^{th} - i)\frac{1}{\lambda_D}\left(\frac{\lambda_D}{\lambda_D + \mu_{sw}}\right)^i \frac{\mu_{sw}}{\lambda_D + \mu_{sw}} \\ &= \frac{1}{\lambda_D}\left[N_D^{th} - \lambda_D E[K_{sw}]\left\{1 - \left(\frac{\lambda_D E[K_{sw}]}{1 + \lambda_D E[K_{sw}]}\right)^{N_D^{th}}\right\}\right]. \end{aligned}$$

Thus, the stability condition for the beneficiary jobs is

$$\begin{aligned} \rho_B &< 1 + \frac{E[R]}{E[L]} = 1 + \frac{\frac{1}{\lambda_D}\left[N_D^{th} - \lambda_D E[K_{sw}]\left\{1 - \left(\frac{\lambda_D E[K_{sw}]}{1+\lambda_D E[K_{sw}]}\right)^{N_D^{th}}\right\}\right]}{\frac{N_D^{th} E[X_D] + E[K_{ba}]}{1-\rho_D} + N_D^{th}\frac{1}{\lambda_D}} \\ &= 1 + \frac{1-\rho_D}{N_D^{th} + \lambda_D E[K_{ba}]}\left[N_D^{th} - \lambda_D E[K_{sw}]\left\{1 - \left(\frac{\lambda_D E[K_{sw}]}{1 + \lambda_D E[K_{sw}]}\right)^{N_D^{th}}\right\}\right] \end{aligned}$$

Note that the right hand side is an increasing function of N_D^{th}; in terms of stability, the larger N_D^{th}, the better the performance. In particular, when $N_D^{th} = 0$, $\rho_B < 1$; as $N_D^{th} \to \infty$, $\rho_B < 2 - \rho_D$. ■

Figure 3 shows the stability condition for beneficiaries as a function of ρ_D. In case (1), we set $E[X_D] = 1$ and $E[K_{sw}] = E[K_{ba}] = 1$. The region below each line satisfies the stability condition. As N_D^{th} increases as high as 100, the effect of switching overhead becomes negligible, and the stability condition approaches $\rho_B < 2 - \rho_D$. In case (2), we set $E[X_D] = 1$ and $E[K_{sw}] = E[K_{ba}] = 10$. The switching cost is large, and it is observed that there is little benefit at moderate or high ρ_D in terms of stability. However, there is still large benefit at low ρ_D. In case (3), we set $E[X_D] = 10$ and $E[K_{sw}] = E[K_{ba}] = 1$. The stability region is much larger; when $N_D^{th} = 1$, the stability region is



(1) $E[X_D] = 1, E[K] = 1$ (2) $E[X_D] = 1, E[K] = 10$ (3) $E[X_D] = 10, E[K] = 1$

Figure 3: *Stability condition on beneficiaries for cycle stealing with switching costs and thresholds.*

almost the same as that of $N_D^{th} = 10$ in case (1). This is intuitive: when $E[X_D] = 10$ and $N_D^{th} = 1$, the expected amount of donor work when the donor switches back is the same as that when $E[X_D] = 1$ and $N_D^{th} = 10$.

5 Validation of analysis

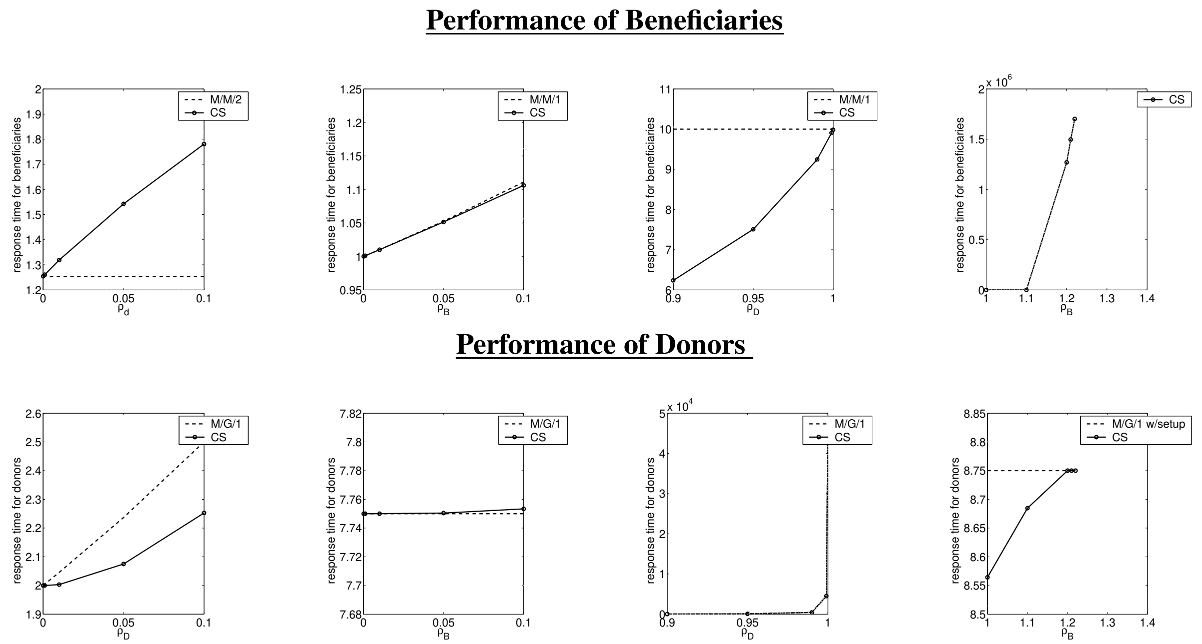
Since our analysis involves approximation of busy periods by 2-stage Coxians, it is of paramount importance to validate the analysis. Analytical validation against limiting cases is presented in Section 5.1, and simulation validation is reported in Section 5.2.

5.1 Validation against known limiting cases

We evaluate the performance of cycle stealing under 4 limiting situations: $\rho_D \to 0$, $\rho_B \to 0$, $\rho_D \to 1$, and $\rho_B \to \rho_B^{max}$, where ρ_B^{max} is the stability condition for ρ_B derived in Section 4. For these limiting cases, the response times of the beneficiaries and donors are easy to evaluate analytically since they converging in performance to either an M/GI/1, an M/GI/1 with setup cost, or an M/M/2. Figure 4 verifies that our analysis of cycle stealing has the correct limiting behavior in all these cases.

5.2 Validation against simulation

The accuracy of our analysis was also validated against simulation: a subset of out validation experiments is shown in Figure 5. Job sizes are assumed to be exponential. We show three cases: In case (a), $E[X_B] = E[X_D] = 1$. In case (b), $E[X_B] = 1$ and $E[X_D] = 10$. In case (c), $E[X_B] = 10$ and $E[X_D] = 1$. In all cases, the results of analysis are in very close agreement with simulation. The only mild discrepancy is the performance of the beneficiaries in case (b), under high load. Under case (b), donor jobs are large, making their busy periods more variable, especially at high loads. As our analysis is very dependent on these busy periods, matching only the first three moments may introduce error in



(1) $\rho_B = 0.9, \rho_D \to 0$ (2) $\rho_B \to 0, \rho_D = 0.6$ (3) $\rho_B = 0.9, \rho_D \to 1$ (4) $\rho_B \to 1.23, \rho_D = 0.6$

Figure 4: *Validation of our cycle stealing analysis against four limiting cases. For lack of space we show graphs only for the specific parameters:* $N_B^{th} = 3$, $E[K_{sw}] = E[K_{ba}] = 1$, X_B *is exponentially-distributed with mean 1 and* X_D *is a Coxian with mean 1 and* $C^2 = 8$.

this case. We hypothesize that the accuracy of our analysis would improve if we matched more moments of the busy periods using Coxians with more stages.

6 Results of Analysis

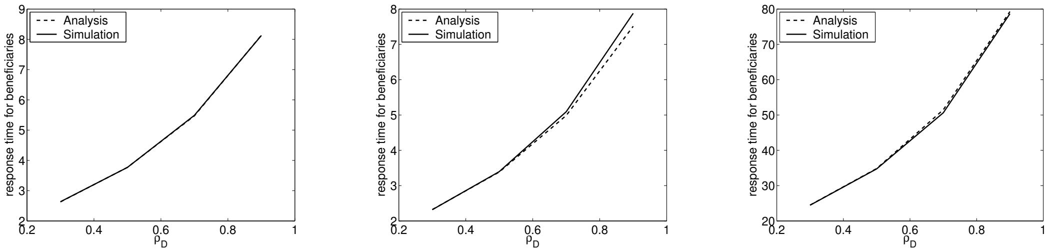
This section discusses our results, organized as take-home messages. We begin with a summary:

6.1 Take-home messages

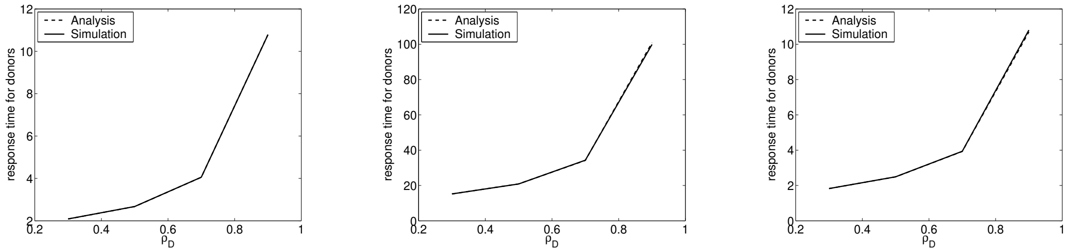
- **(Section 6.2) Cycle stealing is always a win when $\rho_B > 1$, but doesn't pay when $\rho_B < 0.5$.** When $\rho_B > 1$ (and $\rho_D < 1$), cycle stealing can provide an infinite benefit to beneficiaries over dedicated servers, with comparatively little pain to the donors. Recall that the *stability region* for the donors under cycle stealing is much greater than under dedicated servers. Factors such as increased switching costs, increased ρ_D, and increased N_B^{th} result in lower gains for the beneficiary. However since the gain is still infinite these factors are less important in the domain $\rho_B > 1$. When $\rho_B < 0.5$, there is so little gain to the beneficiaries that cycle stealing with non-zero switching overhead doesn't pay.

  We therefore focus the rest of the results section on the remaining case: $0.5 < \rho_B < 1$.

Performance of beneficiaries



Performance of Donors



(a) $E[X_B] = 1, E[X_D] = 1$ (b) $E[X_B] = 1, E[X_D] = 10$ (c) $E[X_B] = 10, E[X_D] = 1$

Figure 5: *Validation of analysis against simulation.* $E[K_{sw}] = E[K_{ba}] = 1$. *We fix* $\rho_S = 0.9$, *and vary* ρ_L. $N_B^{th} = 3$. X_B *and* X_D *are exponentially-distributed.*

- **(Section 6.3) For** $.5 < \rho_B < 1$**, cycle stealing has regions of high gain/low pain and also regions where the reverse is true. These regions depend on job sizes, switching costs, thresholds and loads.** We organize performance into these gain/pain regions. We find that when the switching costs are low, cycle stealing leads to high gain and low pain. However high switching costs can reverse this effect. We find that the performance of the donors is sensitive to the switching cost, while surprisingly, the performance of the beneficiaries is far less sensitive to the switching cost. We also see that higher ρ_B implies higher gains.

- **(Section 6.4) For** $.5 < \rho_B < 1$**, variability of donor job sizes has very little effect.** We find that the variability of the donor jobs has little effect on beneficiary performance. This is very surprising; we expected the beneficiary to gain far less from the bursty help of a donor with irregular (highly variable) job sizes. For $\rho_B > 1$ the above intuition is in fact correct. This echoes findings in [3] and [2] for a different model.

- **(Section 6.5) Setting the best threshold** N_B^{th} **is non-trivial.**

  Finally we tackle the practical problem of how to set N_B^{th}. Higher N_B^{th} implies less gain for the beneficiaries and less pain for the donors. However, the performance of the beneficiaries is relatively insensitive to increasing N_B^{th}, while the donors benefit quite a bit from raising N_B^{th}. The absolute effect of N_B^{th} depends greatly on the particular loads ρ_B and ρ_D. The effect of N_B^{th} on

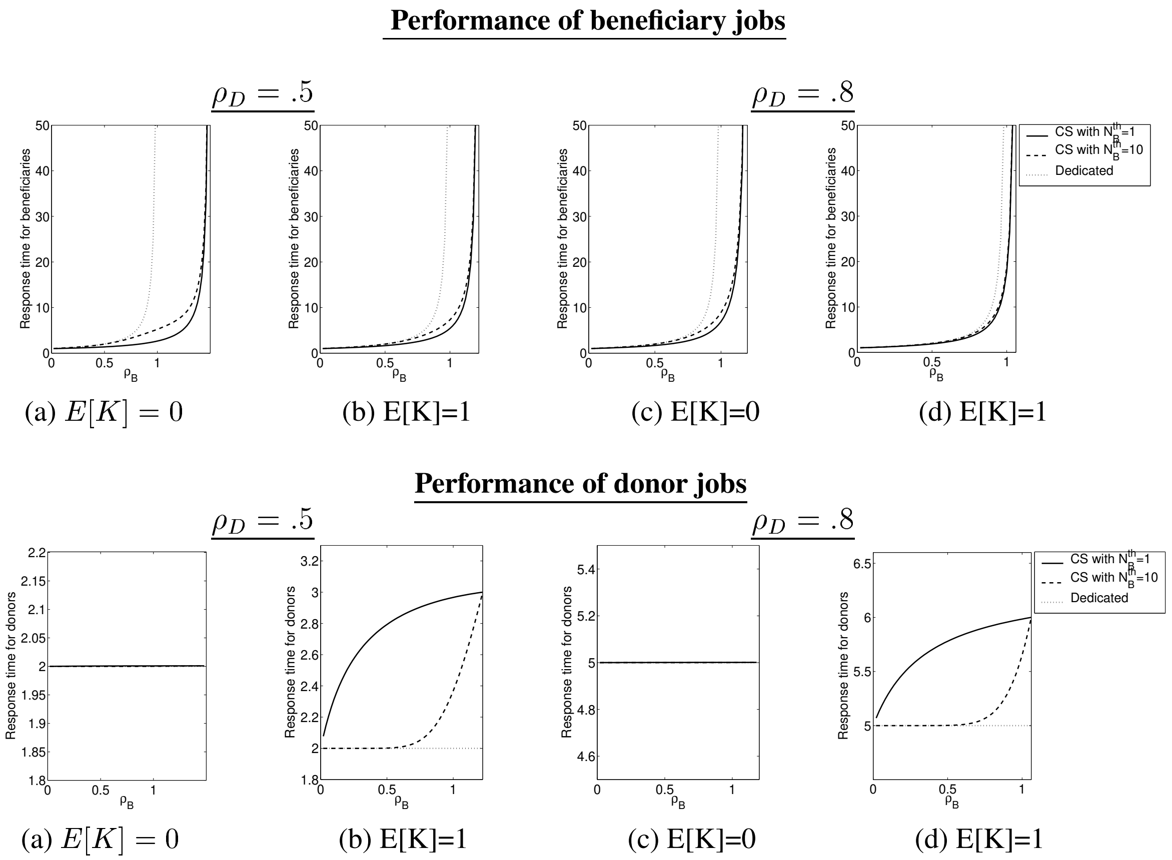


Figure 6: *The response time for beneficiaries and donors as a function of ρ_B. Graphs show the case of (i) cycle stealing with $N_B^{th} = 1$, (ii) cycle stealing with $N_B^{th} = 10$, and (iii) dedicated servers. In all figures X_B and X_D are exponential with mean 1. Switching cost is exponential with mean 0 or 1, as labeled.*

mean performance (of beneficiaries and donors together) is irregular and delicate. In this section we derive five rules of thumb for setting N_B^{th} as a function of the server loads, the switching costs, and the donor job size.

Due to limited space we typically only show a couple options for each parameter. Many more graphs are available in Appendix D.

6.2 Benefits of cycle stealing for wide range of ρ_B

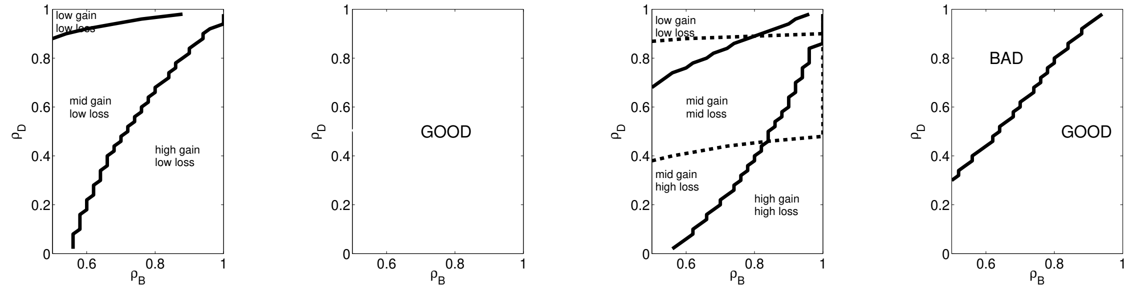
Figure 6 shows the response time for beneficiary jobs (top row) and donor jobs (bottom row) as a function of ρ_B, where ρ_D is held fixed at 0.5 (columns 1 and 2) and at 0.8 (columns 3 and 4). Each graph shows response time under (i) cycle stealing with $N_B^{th} = 1$, (ii) cycle stealing with $N_B^{th} = 10$, and (iii) dedicated servers. The first and third columns have no switching cost, and the second and fourth columns have exponential switching cost with mean 1.

When $\rho_B > 1$, cycle stealing can provide an infinite benefit to beneficiary jobs over dedicated servers. We find that factors like increased switching costs, increased load at donor, and increased N_B^{th} all result in lower gains for the beneficiary. However since the gain is infinite these factors are less important when $\rho_B > 1$. We also see that the response time of donor jobs is bounded by the response time for an M/GI/1 queue with setup cost K, where K is the switch back cost. Increasing N_B^{th} results in less penalty to the donor, converging to the same point for all N_B^{th} values when ρ_B reaches its maximum.

When $\rho_B < 0.5$, the beneficiaries benefit so little that cycle stealing doesn't pay, assuming non-zero switching cost. We therefore focus the rest of the results section on the case: $0.5 < \rho_B < 1$.

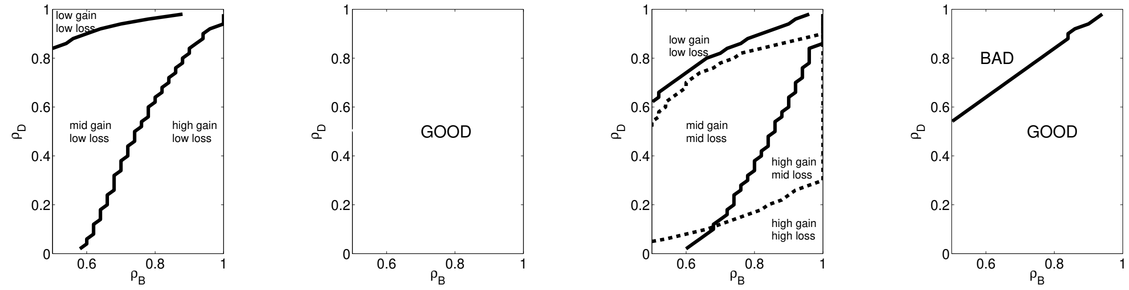
6.3 Benefit of cycle stealing for $.5 < \rho_B < 1.0$

Gain of Beneficiaries and Loss of Donors — $N_B^{th} = 1$



(a) $E[K] = 0$ (b) E[K]=1

Gain of Beneficiaries and Loss of Donors — $N_B^{th} = 3$



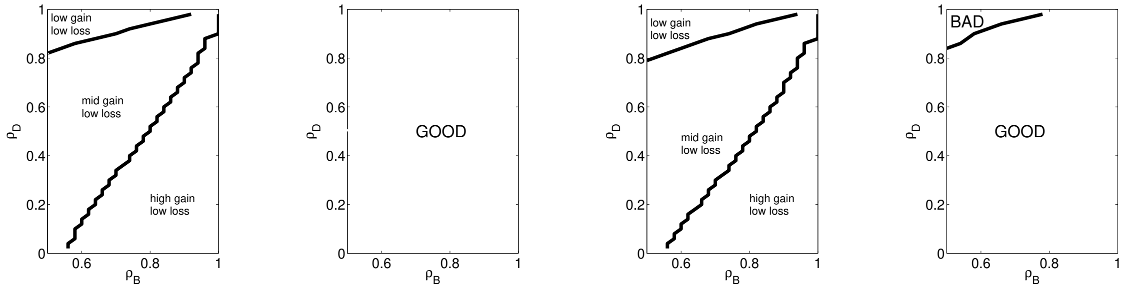
(a) $E[K] = 0$ (b) E[K]=1

Figure 7: *Columns 1 and 3 show the gain of beneficiary jobs and loss of donor jobs relative to dedicated servers. High gain/loss refers to a $>$ 50% difference. Low gain/loss refers to a $<$ 10% difference. Columns 2 and 4 show the mean performance (over all jobs) relative to dedicated servers. In all figures: X_B and X_D are exponentially-distributed with mean 1.*

Figures 7 and 8 evaluate cycle stealing in the region $0.5 < \rho_B < 1.0$, as a function of ρ_D. In Figure 7 X_B and X_D both have mean 1, while Figure 8 assumes X_B has mean 1 and X_D has mean 10. Both figures assume switching cost is 0 or 1, and N_B^{th} is 1 or 3. In our discussion below, we first concentrate on the effect of switching cost, and then look at the effect of N_B^{th}.

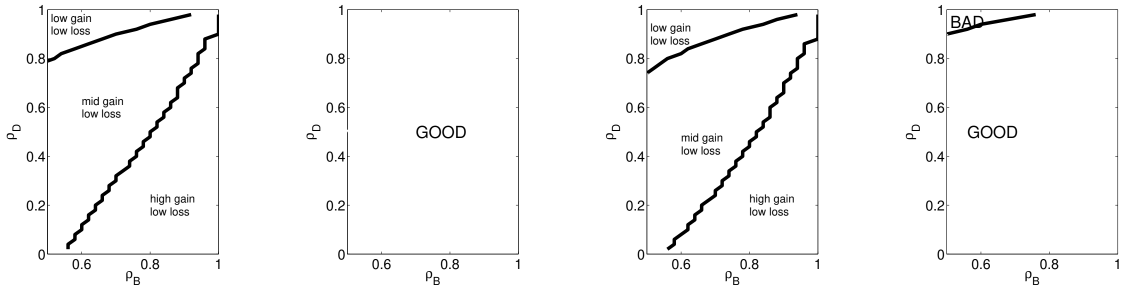
A few definitions are necessary: The term "gain" refers to the improvement in mean response time experienced by beneficiary jobs, as compared with dedicated servers. This gain is expressed as a per-

Gain of Beneficiaries and Loss of Donors — $N_B^{th} = 1$



(a) $E[K] = 0$ (b) E[K]= 1

Gain of Beneficiaries and Loss of Donors — $N_B^{th} = 3$



(a) $E[K] = 0$ (b) E[K]=1

Figure 8: *Columns 1 and 3 show the gain of beneficiary jobs and loss of donor jobs relative to dedicated servers. High gain/loss refers to a $>$ 50% difference. Low gain/loss refers to a $<$ 10% difference. Columns 2 and 4 show the mean performance (over all jobs) relative to dedicated servers. In all figures: X_B and X_D are exponentially-distributed, where X_B has mean 1 and X_D has mean 10.*

centage of the mean response time under dedicated servers. The beneficiary jobs may experience low gain ($< 10\%$), mid gain (between 10% and 50%), or high gain ($> 50\%$). The term "loss" refers to the relative pain (increase) in response time for donor jobs, as compared with dedicated servers. Loss is also expressed as a percentage of the response time under dedicated servers. Odd-numbered columns show various regions of gain and loss. Even-numbered columns use different terms: "good" and "bad." The "good" and "bad" terms refer to mean response time overall (over all jobs including beneficiaries and donors). "Good" indicates that mean response time has dropped as a consequence of cycle stealing. "Bad" indicates that mean response time has increased as a consequence of cycle stealing.

We start by looking at Figure 7. The first and second columns show the case of zero switching cost. We see that all regions are low loss regions (in fact zero loss), and higher ρ_B yields higher gain for the beneficiaries. The third and fourth columns show switching cost of 1. The non-zero switching cost creates slightly reduced gain for the beneficiaries (Recall from Section 6.2 that switching cost has a much more pronounced effect on the beneficiaries when $\rho_B \geq 1$, as the beneficiary is dependent on the donor). More importantly switching cost creates loss for the donor jobs. When ρ_D is very low, the loss appears high, but this is primarily due to the fact that "loss" is defined as a percentage relative to the response time under dedicated servers, and the response time under dedicated servers is obviously low for small ρ_D. The mean gain over all jobs is no longer always positive as shown in the fourth column of Figure 7. Although not shown, we have also investigated higher switching costs, and these lead to the same trend of slightly less gain for beneficiaries and more loss for donors.

Figure 8 differs from Figure 7 only in X_D which now has mean 10. The effect of this change is dramatic: now a switching cost of 1 has almost no effect on either beneficiaries or donors. This makes sense since the setup cost experienced by the donor job is now relatively small compared to its size. We can conclude that cycle stealing is most effective when the switching cost is small relative to the size of the donor jobs.

Consider now columns 2 and 4 of both figures, which depict the effect on overall mean response time. When the switching cost is zero, cycle stealing always overwhelms the dedicated policy. When switching cost is non-zero, cycle stealing is a good idea provided either ρ_B is high, or the switching cost is low compared to X_D. These trends continue in our experiments with higher switching costs.

Finally, we discuss the effect of N_B^{th} in both figures. The performance of the beneficiaries is relatively insensitive to increasing N_B^{th} from 1 to 3 in both figures; however the donors benefit quite a bit from raising N_B^{th}. We further investigate selection of N_B^{th} in Section 6.5.

6.4 Effect of donor job size variability

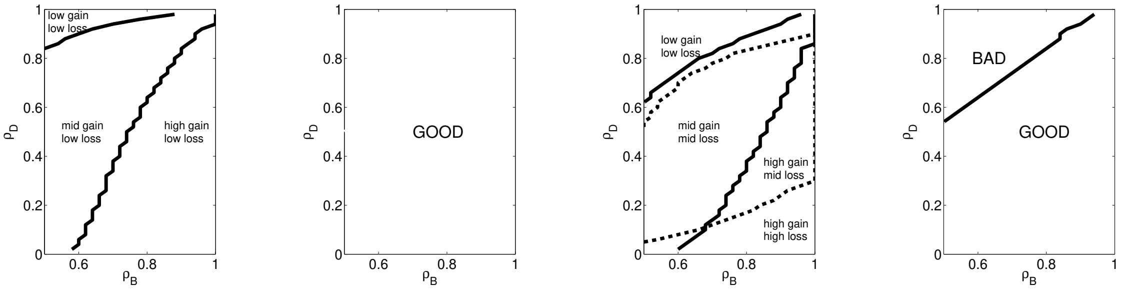
In this section, we consider the effect of variability in the donor job sizes. It seems intuitive that when donor job sizes are made more variable, two things should happen:

1. The donor pain percentage should go down. This is because the donor response times will be higher overall and so the relative pain will appear to be less, and
2. The beneficiary gain should go down. This is because high variability in the donor job sizes, implies high variability in the length of the donor busy periods, which implies that the donor's visits to the beneficiary queue will become more irregular. Sporadic help should be inferior to regular help, for the beneficiary.

Figure 9 below shows that the first hypothesis above is in fact true, while the second hypothesis is surprisingly false, at least for $\rho_B < 1$. Comparing row 1 (X_D has low variability: $C^2 = 1$) with row 2 (X_D has high variability: $C^2 = 8$) we see that there is no discernible difference in donor performance.

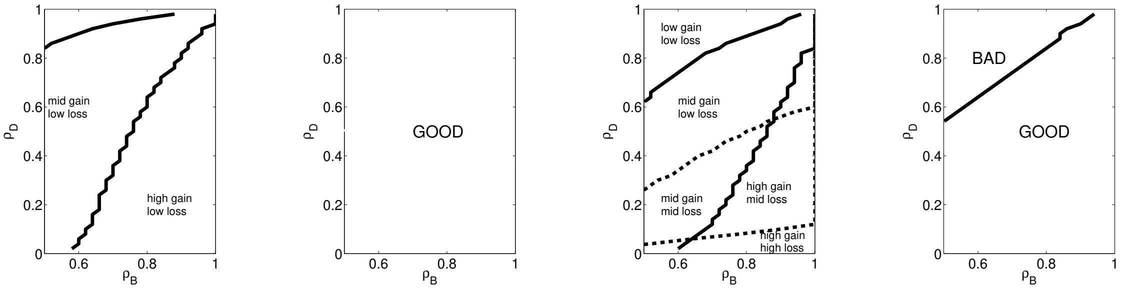
To study this effect more closely, we next increase the variability in the donor job sizes to $C^2 = 50$. Figure 10 shows the performance of just the beneficiary jobs under the case of zero switching cost, when the donor job size variability is 1, 8, or 50, and ρ_D is fixed at 0.5. ρ_B is allowed to vary from 0 to the

Gain of Beneficiaries and Loss of Donors — X_D with $C^2 = 1$



(a) $E[K] = 0$ (b) $E[K] = 1$

Gain of Beneficiaries and Loss of Donors — X_D with $C^2 = 8$



(a) $E[K] = 0$ (b) $E[K] = 1$

Figure 9: *Beneficiary gain and donor pain shown for donor job sizes with low variability (top) and high variability (bottom). Under higher donor variability, percentage of donor pain is lessened, but percentage of beneficiary gain stays the same. All graphs assume: X_B and X_D have mean 1, $N_B^{th} = 3$.*

stability condition. As is observed in Figure 9, the effect of variability of X_D on the performance of X_B is very small when $\rho_B < 1$. When $\rho_B > 1$, the effect of variability in donor sizes is non-negligible, but is still not as great as one might expect.

6.5 Optimal setting for beneficiary threshold, N_B^{th}

This section investigates the optimal setting for N_B^{th}. We start with some obvious rules: (i) increasing N_B^{th} leads to lower gain for the beneficiaries and lower pain for the donors; (ii) if the switching cost is zero, the optimal N_B^{th} is 1, since there is never any pain for the donors.

Figure 11 shows optimal values of N_B^{th} for minimizing mean response time over all jobs (donors and beneficiaries) as a function of ρ_B and ρ_D under various switching costs and donor job sizes. The numbers on the contour curves show the best N_B^{th} at each load. For clarity we only show lines up to $N_B^{th} = 14$. The following additional rules of thumb are implied by the figure: (iii) the optimal N_B^{th} is an increasing function of ρ_D and a decreasing function of ρ_B; (iv) increasing the switching cost increases the optimal N_B^{th}; (v) increasing the donor job size leads to lower optimal N_B^{th}.

Effect of variability of donor jobs on performance of beneficiary jobs

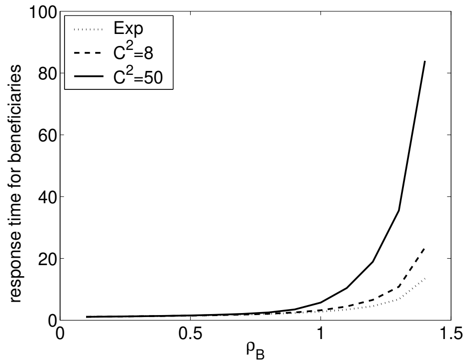
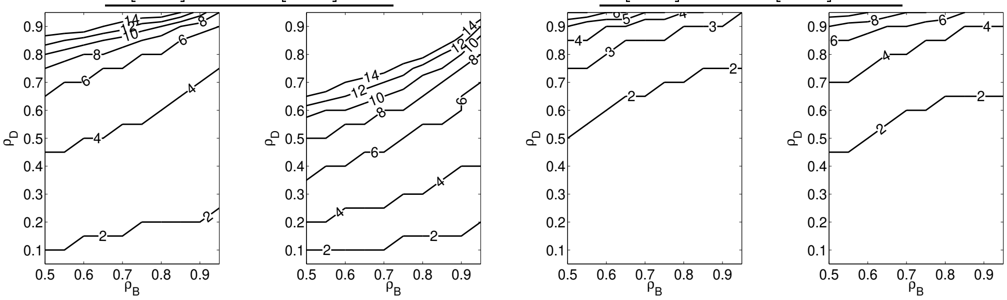


Figure 10: *The response time for beneficiary jobs under different donor job size variability. ρ_D is fixed at 0.5. ρ_B varies from 0 to the stability condition of 1.5. The mean donor job size and beneficiary job size is 1, and switching cost is zero.*

Selecting the optimal N_B^{th}

$E[X_B] = 1, E[X_D] = 1$ $\qquad$ $E[X_B] = 1, E[X_D] = 10$



E[K]=1 $\qquad$ E[K]=2 $\qquad$ E[K]=1 $\qquad$ E[K]=2

Figure 11: *Graphs showing the optimal value of N_B^{th} with respect to mean response time over all jobs.*

7 Extensions and Current Work

This paper solves the problem of cycle stealing with switching costs and thresholds, presenting many insights into the characteristics and performance of cycle stealing. Our analytical approach easily generalizes to more complex cycle stealing models. For example, in this paper we have assumed that the donor immediately switches back when a donor job arrives. As we've stated earlier, we can also solve the more general case where the donor only switches back when N_D^{th} donor jobs are waiting. Furthermore, we don't even need to require that the donor switches back immediately at this point; we can allow the donor to first complete the beneficiary job in progress. Completing the beneficiary job obviates the need for checkpointing the job, however it sometimes reduces system performance, particularly when beneficiary jobs have higher variability than donor jobs. We can also handle the problem of one beneficiary and two or more donors, where if i donors are helping, the beneficiary sees an $M/GI/i$ queue. This extension also allows the different donors to have different switching thresholds and switching costs.

References

[1] S. Asmussen. Phase-type distributions and related point processes: Fitting and recent advances. In S. R. Chakravarthy and A. S. Alfa, editors, *Matrix-Analytic Methods in Stochastic Models*, pages 137–149. Marcel Dekker, 1997.

[2] S. Borst, O. Boxma, and P. Jelenkovic. Reduced-load equivalence and induced burstiness in gps queues with long-tailed traffic flows. *Under submission*.

[3] S. Borst, O. Boxma, and M. van Uitert. The asymptotic workload behavior of two coupled queues. *Under submission*.

[4] J. Cohen and O. Boxma. *Boundary Value Problems in Queueing System Analysis*. North-Holland Publ. Cy., 1983.

[5] G. Fayolle and R. Iasnogorodski. Two couple processors: the reduction to a riemann-hilbert problem. *Zeitschrift fur Wahrscheinlichkeitstheorie und vervandte Gebiete*, 47:325–351, 1979.

[6] A. Konheim, I. Meilijson, and A. Melkman. Processor-sharing of two parallel lines. *Journal of Applied Probability*, 18:952–956, 1981.

[7] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, Philadelphia, 1999.

[8] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models*. Johns Hopkins University Press, 1981.

[9] T. Osogami and M. Harchol-Balter. Matching three moments of distributions and busy periods by n-stage coxians. Technical Report CMU-CS-02-178, School of Computer Science, Carnegie Mellon University, September 2002.

[10] H. Takagi. *Queueing Analysis – A Foundation of Performance Evaluation*, volume 1: Vacation and Priority Systems, Part 1. North Holland, 1991.

A Analysis of a variant: Donor switches back when unneeded

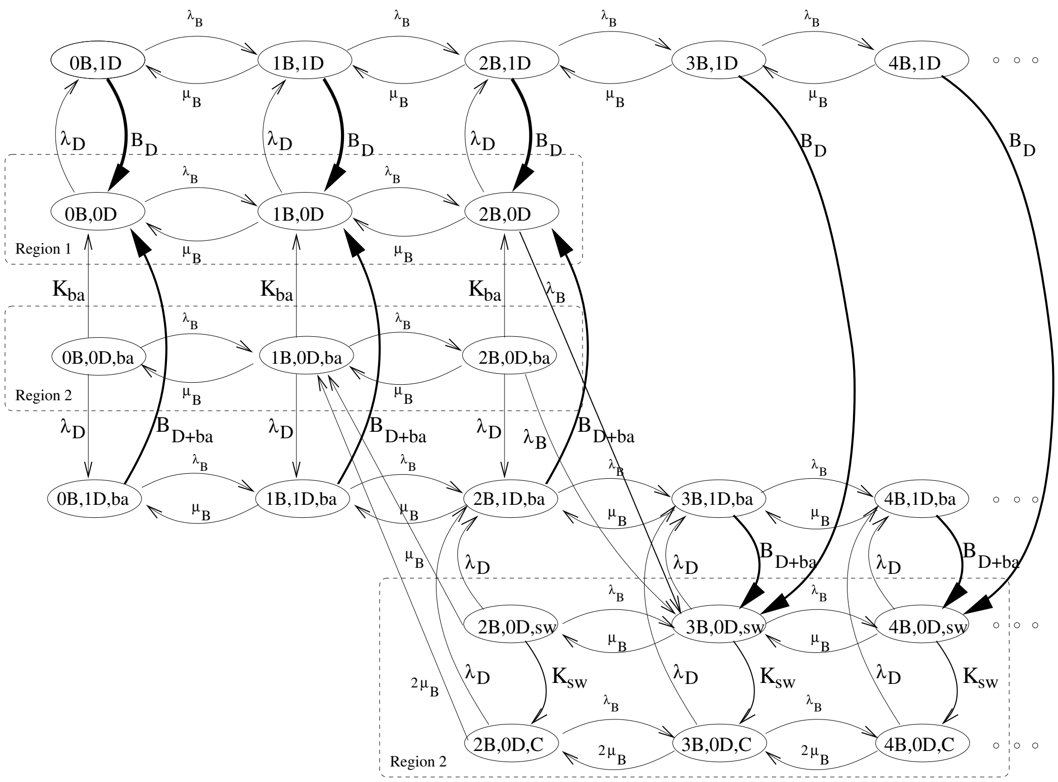


Figure 12: *Markov chain of a variant. The donor server switches back when there is at most one beneficiary job regardless of the number of donor jobs.*

In this section, we consider a variant of our algorithm: the donor server switches back when there is ≤ 1 beneficiary job, even if there are no donor jobs at the time. One might imagine that the performance of donor jobs would improve under this policy, especially when the arrival rate of donor jobs is high compared to that of beneficiary jobs, and the cost of switching back is relatively high.

Figure 12 shows the Markov chain for the analysis of the variant in the case of $N_B^{th} = 3$. It is assumed that if the beneficiary server becomes idle while the donor server is still working on a beneficiary job, then the job in progress at the donor server is passed over to the beneficiary server without any cost. It is also assumed that K_{sw}, K_{ba}, and X_B are exponentially distributed, all of which can be generalized by using Coxian distributions. The response time of the donor job is analyzed in the same way as before, except that Region 2 in the chain has been expanded.

Having analyzed the above Markov chain, we find that counter to our intuition this variant does not significantly affect the response time for the donor or the beneficiary, under all conditions studied.

B Incorporating a Donor threshold N_D^{th}.

Throughout the paper we have assumed that the donor server switches back immediately when a new donor job arrives at an empty donor queue. However, this is not necessarily the best policy with respect to overall mean performance; given high switching costs, it might be better for the donor server to only switch back if there are at least $N_D^{th} > 1$ donor jobs waiting. Furthermore, in Section 4 we found that the stability region for the beneficiary is an increasing function of N_D^{th}, thereby giving further justification for raising N_D^{th}. In this section, we show how to analyze a generalization of our algorithm, which assumes general N_D^{th}.

Figure 13 shows a Markov chain that corresponds to cycle stealing where $N_D^{th} = 2$ and $N_B^{th} = 3$.

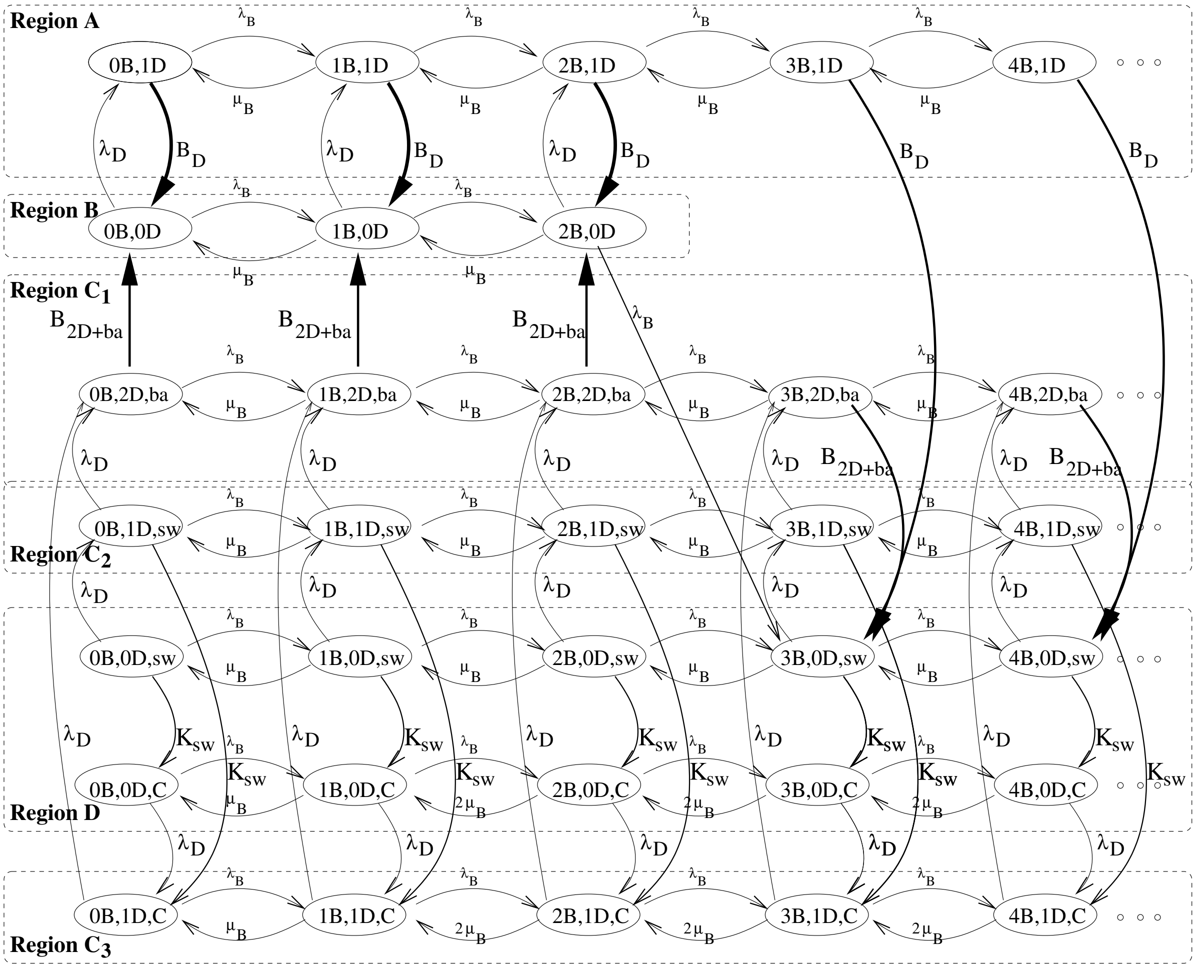


Figure 13: *Markov chain for cycle stealing with $N_D^{th} = 2$ and $N_B^{th} = 3$ and exponentially-distributed X_B and K_{sw}.*

B.1 Response time for donor jobs

We first analyze the response time for donor jobs when $N_D^{th} = 2$. Then, the results are extended to general N_D^{th}.

The states of the Markov chain are divided into four regions A, B, C, and D, where Region C is further subdivided into three regions C_1, C_2, and C_3, as shown in Figure 13. The time in queue for a donor arrival is analyzed by conditioning on the region it arrives into. That is, the expected time in queue is

$$E[T_Q] = \sum_{i:\text{all regions}} E[T_Q|\text{Region } i]\Pr(\text{arrival sees Region } i),$$

where $E[T_Q|\text{Region } i]$ is the expected time in queue given that the arrival sees Region i.

When a donor job arrives in Region A, it sees a busy M/GI/1 queue, where the job sizes $\sim X_D$ and the arrival rate is λ_D. Recall that the expected time in queue for an M/GI/1 is

$$E[T_Q^{M/GI/1}] = \frac{\rho_D}{1-\rho_D}\frac{E[X_D^2]}{2E[X_D]}$$

which is also written as

$$E[T_Q^{M/GI/1}] = E[T_Q^{M/GI/1}|busy]Pr(busy) + E[T_Q^{M/GI/1}|idle]Pr(idle)$$

where $E[T_Q^{M/GI/1}|busy]$ is the expected time in queue in an M/GI/1 given that the arrival finds the system busy. Observing that $E[T_Q^{M/GI/1}|idle] = 0$, and the probability that the system is busy is $Pr(busy) = \rho_D = \lambda_D E[X_D]$, we have

$$E[T_Q|\text{Region A}] = E[T_Q^{M/GI/1}|busy] = \frac{1}{1-\rho_D}\frac{E[X_D^2]}{2E[X_D]}$$

When a donor job arrives in Region B, the waiting time is zero, since the donor server is staying at an empty donor queue. Thus,

$$E[T_Q|\text{Region B}] = 0$$

To analyze the performance of a donor job, given that it arrives in Region C, first note that if we aggregate Regions C_2 and C_3 into $C_{2,3}$, the duration of staying at Region $C_{2,3}$ is exponentially distributed with rate λ_D, and the duration of staying at Region C_1 is the length of a busy period started by a job of size X_D and a setup time $S = K_{ba} + X_D$. Furthermore, Region C_1 and Region $C_{2,3}$ are visited alternately; Regions A, B, and D are visited on the way from Region C_1 to Region $C_{2,3}$ (see Figure 14). Therefore, the fraction of time that the system spends in Region C_1 given that the system is either in Region C_1 or in Region $C_{2,3}$ is the same as the fraction of time that the M/GI/1 queue with setup time S

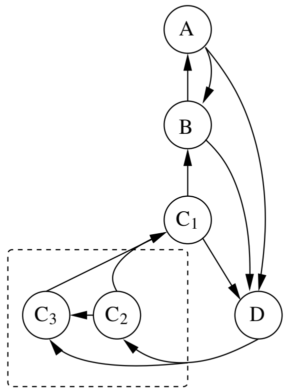


Figure 14: *Transitions among regions of Markov chain in figure 13*

is busy. The expected time in queue given that an arrival sees Region C_1 ($C_{2,3}$, respectively) equals the expected time in queue for an M/GI/1 queue with setup time, given that the arrival finds the system busy (idle, respectively). Therefore, the expected time in queue given that an arrival sees Region C is just the expected time in queue in a corresponding M/GI/1 queue with setup time S:

$$\begin{aligned} E[T_Q|\text{Region C}] &= E[T_Q^{M/GI/1 \text{ with setup } S}] \\ &= \frac{2E[S] + \lambda_D E[S^2]}{2(1+\lambda_D E[S])} + \frac{\lambda_D E[X_D^2]}{2(1-\rho_D)} \end{aligned}$$

When a donor job arrives in Region D, it waits for one more donor job to arrive, after which the donor server switches back. Thus, the expected time in queue of an arrival into Region D is

$$E[T_Q|\text{Region D}] = \frac{1}{\lambda_D} + E[K_{ba}]$$

In summary, the time in queue for donor jobs is

$$\begin{aligned} E[T_Q] &= E[T_Q^{M/GI/1}|busy]\Pr(\text{Region } A) + E[T_Q^{M/GI/1 \; with \; setup \; S}]\Pr(\text{Region } C) \\ &+ \left(\frac{1}{\lambda_D} + E[X_D]\right)\Pr(\text{Region } D) \end{aligned}$$

The analysis is easily extended to general N_D^{th}. The Markov chain describing the states of such a system can still be divided into four regions A, B, C, and D, but Region D is further divided into $n-1$ regions D_1, ..., D_{n-1} (Region C is still subdivided into three regions, as before). In Region A, the donor server is working on donor jobs. In Region B, the donor server is idle at the donor queue. In Region C_1, the donor server is in the process of switching back to the donor queue. In Region C_2, the donor server is in the process of switching to the beneficiary queue and there are $n-1$ donor jobs. In Region C_3, the donor server is working on beneficiary jobs and there are $n-1$ donor jobs. In Region D_i, the donor server is either switching to the beneficiary queue or working on beneficiary jobs and there are $i-1$ donor jobs, for $1 \leq i \leq n-1$. A similar argument as above leads to the time in queue for donor jobs:

$$\begin{aligned} E[T_Q] &= E[T_Q^{M/GI/1}|busy]\Pr(\text{Region } A) + E[T_Q^{M/GI/1 \text{ } with \text{ } setup \text{ } S}]\Pr(\text{Region } C) \\ &\quad + \sum_{i=1}^{n-1} \left(\frac{1}{\lambda_D} + (n-1)E[X_D] \right) \Pr(\text{Region } D_i), \end{aligned}$$

where the setup time is now $S = K_{ba} + \sum_{i=1}^{n-1} X_D$.

C Superposition of the chains

Figure 15 shows a superposition of the chains in Figure 1(b) and (c).

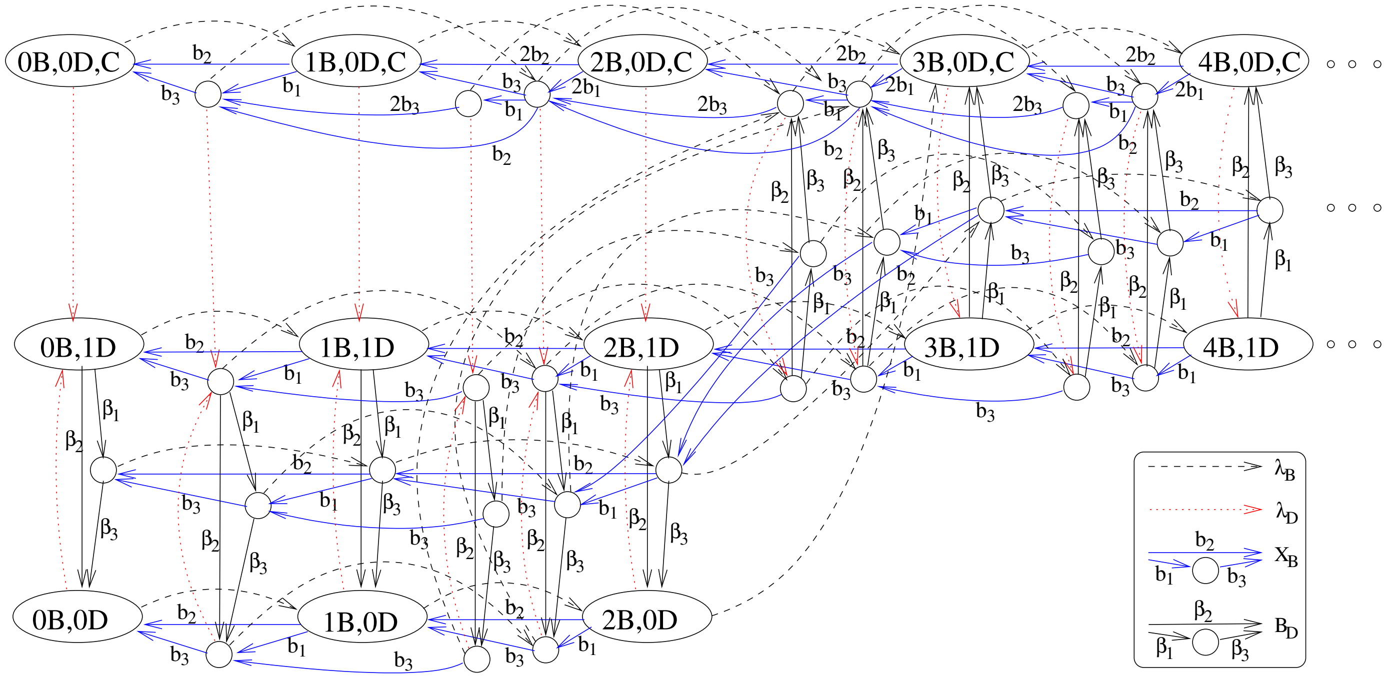


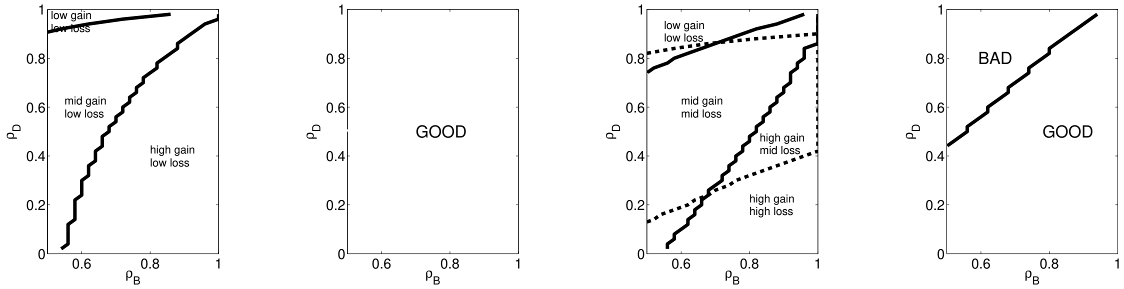
Figure 15: *Markov chain for cycle stealing without switching cost, where X_B and B_D are both represented by 2-stage Coxians.*

D More graphs

In this section we include a few more result graphs omitted from the body of the paper for lack of space. Figure 16 shows the gain/loss performance regions from Section 6.3 for the case where the beneficiaries now have mean size 10 and the donors have mean size 1. The performance of the beneficiaries is virtually identical to the case where X_D and X_B have mean 1. The donor performance is improved. The reason for the improvement in donor performance is that the donor visits the beneficiary less frequently, since the number of jobs queued at the beneficiary isn't as high, for large jobs.

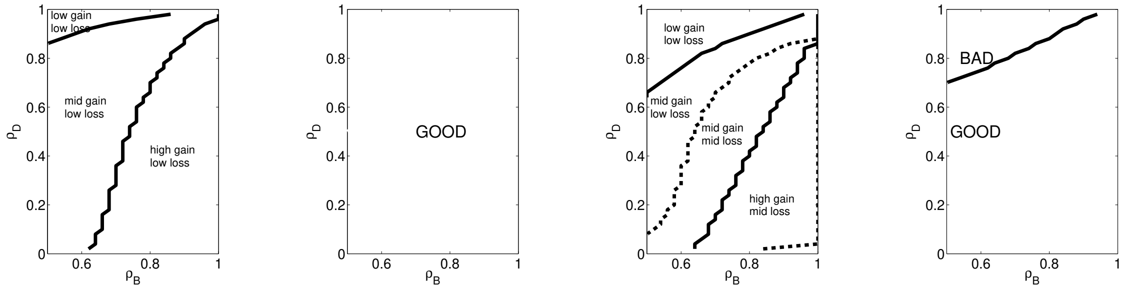
Figure 17 shows additional optimal N_B^{th} computations, for the case of large beneficiary jobs and the case of highly variable donor jobs. The case of large beneficiary jobs, results in lower optimal N_B^{th}. Increasing the variability of donor jobs doesn't seem to have any effect.

Gain of Beneficiaries and Loss of Donors — $N_B^{th} = 1$



(a) $E[K] = 0$ (b) E[K]= 1

Gain of Beneficiaries and Loss of Donors — $N_B^{th} = 3$

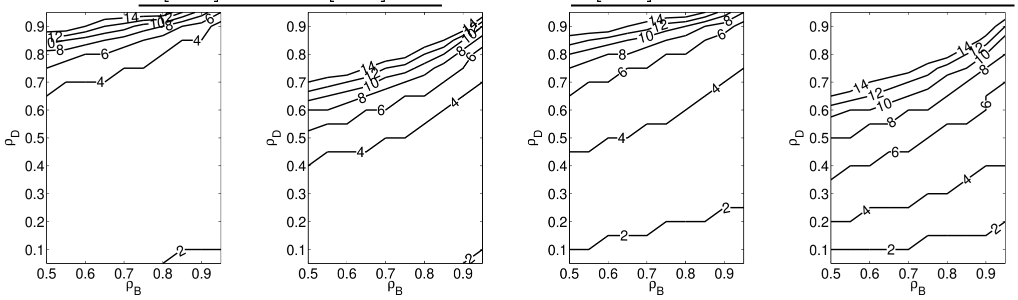


(a) $E[K] = 0$ (b) E[K]=1

Figure 16: *Columns 1 and 3 show the gain of beneficiary jobs and loss of donor jobs relative to dedicated servers. High gain/loss refers to a $> 50\%$ difference. Low gain/loss refers to a $< 10\%$ difference. Columns 2 and 4 show the mean performance (over all jobs) relative to dedicated servers. In all figures, X_B and X_D are exponentially-distributed, where X_B has mean 10 and X_D has mean 1.*

Selecting the optimal N_B^{th}

$E[X_B] = 10$, $E[X_D] = 1$ $\qquad$ $E[X_B] = 1$, X_D: mean 1 and $C^2 = 8$



E[K]=1 $\qquad$ E[K]=2 $\qquad$ E[K]=1 $\qquad$ E[K]=2

Figure 17: *Graphs showing the optimal value of N_B^{th} with respect to mean response time over all jobs.*