Analysis of Cycle Stealing with Switching Cost

Takayuki Osogami
CSD
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
osogami@cs.cmu.edu

Mor Harchol-Balter*
CSD
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
harchol@cs.cmu.edu

Alan Scheller-Wolf
GSIA
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
awolf@andrew.cmu.edu

ABSTRACT

We consider the scenario of two 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 accurate as desired, and 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.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques

General Terms

Performance, Algorithms

Keywords

Cycle stealing, task assignment, load sharing, server farm, distributed system, supercomputing, matrix analytic, starvation, unfairness

*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.

SIGMETRICS'03, June 10–14, 2003, San Diego, California, USA.

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 the machine of a different user, Dan, 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 secondary goal is to derive parameter settings for cycle stealing. In particular, given non-zero switching costs, cycle stealing may pay only 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 there are two queues, the *beneficiary* queue and the *donor* queue, with independent arrival processes and service time distributions operating as M/GI/1/FCFS queues. 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 three moments of the service times are finite, and queues are stable. The donor server cannot work on any job while it is in the switching or switching back state.

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 [10]. 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] Third, 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 the associated technical report [10], 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 thresholds are zero. Even for this simplest instance the continuous-time Markov chain, while easy to describe, appears computationally 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 easily exploited in practice 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 switching 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 the other processor. This help incurs no switching cost and has a benefit if even a single job is present. These models inherently assume a preemptive resume discipline: when a processor returns to its own queue, none of its work is lost. 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 the single queue for each class. All works we mention below consider Poisson arrivals.

Early work on the coupled processor model was by Fayolle and Iasnogorodski [4] 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). While it is possible to numerically evaluate these analytical expressions, they were not evaluated in either work, thus no intuition was provided on the performance of these systems.

The above work was extended by Cohen and Boxma [3] 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, but not for response time. 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 [2] apply a transform method to the expressions in [3], 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 [1] 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 noncooperative load is larger than one. 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 thresholds. Recall that the difficulty in analyzing cycle stealing is that the corresponding stochastic process defines a

[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. 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 (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 job sizes more variable than exponential, the error increases.

Markov chain which grows infinitely in two dimensions (2D), making it computationally 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 of seconds are required to generate most 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 accurate 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 each class are assumed to be drawn i.i.d. from general distributions (which we approximate by a Coxian). The arrival process can easily be generalized to a MAP – Markovian Arrival Process [7].

Summary of results

Our analysis yields many interesting results concerning cycle stealing, detailed in Sections 4 and 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 thus is very bursty, the beneficiaries still enjoy significant benefits.

The effect of the thresholds, N_B^{th} and N_D^{th}, is also interesting: Increasing N_B^{th} curbs the beneficiary gain only slightly, but cuts the donor pain significantly. N_B^{th} does not affect the stability region for either donor or beneficiary. By contrast, increasing N_D^{th} increases the stability region of the beneficiaries (thus providing unbounded gain for the beneficiaries), while only increasing donor pain by a finite amount (the donor stability region is not affected). We describe the optimal choice of N_B^{th}, under a range of server loads, threshold levels, and switching costs.

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 servers. 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 SWITCH 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 alternately the case of generally-distributed (Coxian) job sizes or busy periods. The actual chain that we evaluate in the paper is the superposition of the chains in Figures 1(b) and (c) where job sizes and busy periods are Coxian, see [10].

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*: indicating that the donor processor is cooperating with the beneficiary; (ii) *independent row, with* ≥ 1 *donor job*: indicating that the donor and beneficiary processors are each at their own queues and there is at least 1 donor job present; (iii) *independent row, with* 0 *donor jobs*: indicating 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 empty. 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 beneficiary job arrives while we are in this row and the number beneficiaries is at least N_B^{th}, then we transition to the cooperating row. If a donor job arrives while we're in the row where the processors are working independently with zero donor jobs, 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 donor busy period the number of beneficiaries 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 at the end of the donor busy period the number of beneficiaries is at least N_B^{th}, then we move to the cooperating row, where we stay until there is a donor arrival. Note that the thresh-

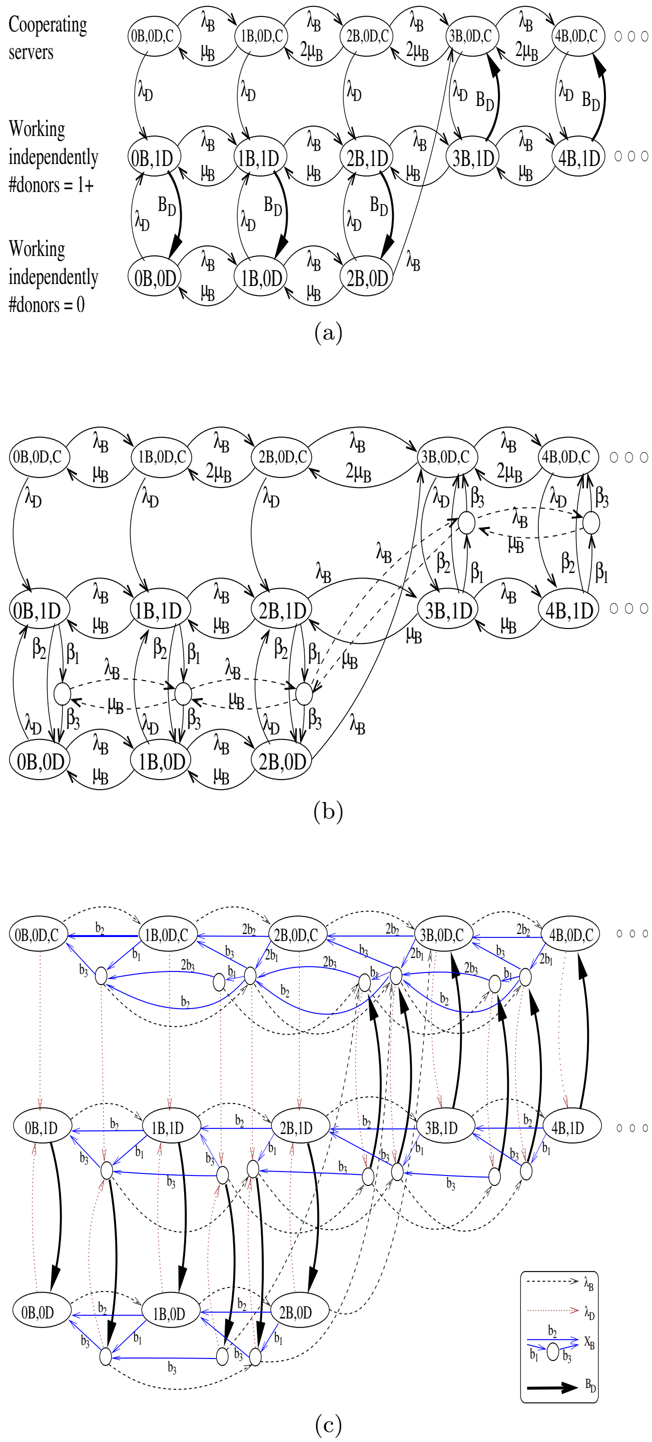


Figure 1: Markov chain for cycle stealing without switching cost where $N_B^{th} = 3$. (a) Where X_B is exponential and B_D is drawn using a single (exponential) transition. (b) Where X_B is exponential, and B_D is represented by a 2-stage Coxian. (c) Where X_B is represented by a 2-stage Coxian and B_D is drawn using a single (exponential) transition.

old N_B^{th} is used only to decide about the transition from residing at separate queues to cooperating; the donor server keeps working on the beneficiary jobs even when the number of beneficiaries is $< N_B^{th}$, until a new donor job arrives.

Observe in Figure 1(c) that when the donor and beneficiary are cooperating, the state space now must maintain states for one or two partially-completed beneficiary jobs. If a donor job arrives while we're in the cooperating row, the 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 conditions 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 [7, 8]. This is a simple, compact and efficient method for solving QBD (quasi-birth-death) Markov chains which are infinite in one dimension, where the chain repeats itself after some point, as does Figure 1. Applying Little's Law then yields their mean response time. Most of the plots in this paper which uses matrix analytic methods was produced within a couple of seconds using Matlab 6 running on Linux, on a 1 GHz Pentium III with 512 MB RAM.

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 empty 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 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

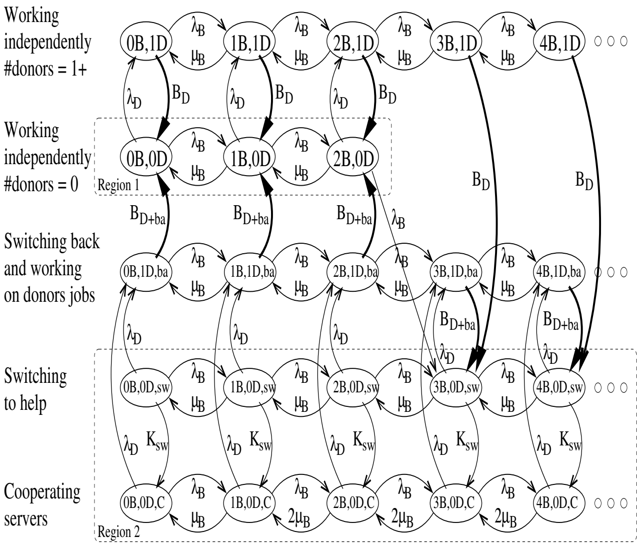


Figure 2: Markov chain for cycle stealing with switching cost; X_B and K_{sw} are exponential, $N_B^{th} = 3$.

row, as soon as there are N_B^{th} beneficiary jobs. After K_{sw} time, if no donor job has 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}, which we denote by B_{D+ba}.

In our Markov chain, we have again drawn the busy period transitions with a single bold transition, although in evaluating the chain we will replace this bold transition with a minimal Coxian that matches the first three moments of the busy period duration. Computing the first three moments of the busy period, B_{D+ba}, is straightforward from the Laplace transform below:

$$\widetilde{B_{D+ba}}(s) = \widetilde{K_{ba}}(s+\lambda_D-\lambda_D\widetilde{B_D}(s))\cdot\widetilde{X_D}(s+\lambda_D-\lambda_D\widetilde{B_D}(s)).$$

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:

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

where region 1 and region 2 are defined in Figure 2. 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 [12]:

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

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} (note: throughout this paper we assume $N_D^{th} = 1$; however, in this section we consider general N_D^{th}).

The stability conditions are not always intuitive. We find for example that the stability condition for the donor jobs remains $\rho_D < 1$, regardless of the fact that the donor jobs experience switching costs. By contrast, the stability condition for the beneficiary jobs is a function which can be significantly below $2 - \rho_D$, specifically because the switching cost eats at the stability region. Also, interestingly, the value of N_B^{th} does not affect the stability region of either the donor or beneficiary jobs. By contrast, increasing N_D^{th} increases the stability region of the beneficiary jobs; however it has no effect of the stability region of the donor jobs.

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 $\rho_D \geq 1$. $\square$

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

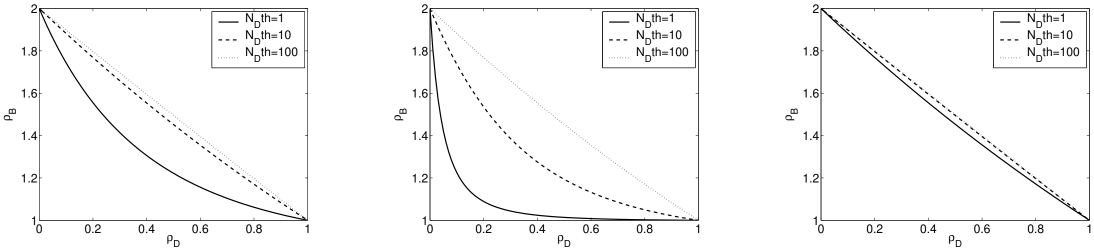
LEMMA 1. *If the beneficiary queue is stable at $N_B^{th} = 0$, then it is stable at $N_B^{th} = n$, $\forall\ 0 \leq n < \infty$.*

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 the number of jobs at time $t = 0$ is n, instead of 0 as in the original process, and 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 $\widehat{L}_B^{(n)}(t)$ and hence $L_B^{(n)}(t)$. $\square$

We now prove the stability condition on ρ_B.

THEOREM 2. *The stability condition (necessary and sufficient) for beneficiary jobs with donor threshold N_D^{th} is:*

$$\rho_B < 1 + \frac{\max(1-\rho_D, 0)\sum_{i=0}^{N_D^{th}} (N_D^{th} - i)\frac{(-\lambda_D)^i}{i!}\tilde{K}_{sw}^{(i)}(\lambda_D)}{N_D^{th} + \lambda_D E[K_{ba}]},$$



(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.

where $\tilde{K}_{sw}(s)$ is the Laplace transform of K_{sw} and $\tilde{K}_{sw}^{(i)}(s)$ is its i-th derivative. In particular, when K_{sw} is exponentially distributed, the condition is expressed by the closed form:

$$\rho_B < 1 + \frac{\max(1-\rho_D, 0)\left\{N_D^{th} - \frac{q}{1-q}\left(1 - q^{N_D^{th}}\right)\right\}}{N_D^{th} + \lambda_D E[K_{ba}]},$$

where $q = \frac{\lambda_D E[K_{sw}]}{1+\lambda_D E[K_{sw}]}$.

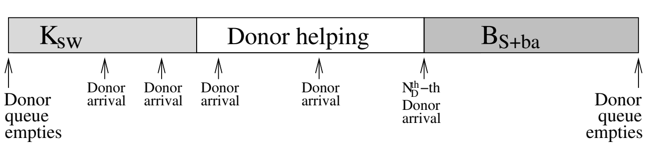
Proof. We first prove sufficiency. By Lemma 1, we are allowed to assume $N_B^{th} = 0$. Let F denote the time average fraction of time that the donor helps the beneficiary. Then the strong law of large numbers can be used to show that the necessary and sufficient condition for stability of the beneficiary 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 queue becomes idle. Recall $N_B^{th} = 0$. By renewal-reward theory:

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

where R denotes the help (reward) during the renewal cycle, and Y 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 and B_{S+ba} denote the length of the busy period started by these jobs of total size S plus K_{ba}. Then, as $\rho_D < 1$,

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

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, $(N_D^{th} - i)E[I_D]$.

Let p_i denote the probability that there are i donor arrivals during K_{sw}. Then, $p_i = \frac{(-\lambda_D)^i}{i!} \tilde{K}_{sw}^{(i)}(\lambda_D)$. Using p_i, $E[R]$ is now derived as follows:

$$E[R] = \sum_{i=0}^{N_D^{th}-1} (N_D^{th} - i) \frac{1}{\lambda_D} p_i.$$

When K_{sw} is exponentially-distributed, $p_i = q^i(1-q)$, where $q = \frac{\lambda_D}{\lambda_D + \mu_{sw}}$. Therefore,

$$E[R] = \frac{1}{\lambda_D}\left\{N_D^{th} - \frac{q}{1-q}(1 - q^{N_D^{th}})\right\}.$$

The stability condition for the beneficiary jobs is thus

$$\rho_B < 1 + \frac{E[R]}{E[Y]} = 1 + \frac{\sum_{i=0}^{N_D^{th}-1}(N_D^{th} - i)p_i}{N_D^{th} + \lambda_D E[K_{ba}]},$$

where $p_i = \frac{(-\lambda_D)^i}{i!} \tilde{K}_{sw}^{(i)}(\lambda_D)$. In particular, when K_{sw} is exponentially distributed, the condition becomes:

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

Above, we have proved the necessary and sufficient condition for $N_B^{th} = 0$. By Lemma 1, this is also the sufficient condition for $N_B^{th} > 0$. Now, we prove necessity for $N_B^{th} > 0$. When $N_B^{th} > 0$, the donor server does not necessarily help the beneficiary even when it is available for help. Therefore, there are two types of renewal periods. In the first type of renewal period, the donor server helps the beneficiary, i.e. $R > 0$. In this case, $E[Y]$ is the same as for $N_B^{th} = 0$, and $E[R]$ for $N_B^{th} > 0$ is at most $E[R]$ for $N_B^{th} = 0$. In the second type of renewal period, the donor server does not help the beneficiary, i.e. $R = 0$. (In this case $E[Y]$ can be smaller than that for $N_B^{th} = 0$.) The fraction of time, F, that the donor server helps the beneficiary can be expressed as $F = F_1 + F_2$, where F_1 is the fraction of time that the donor server helps the beneficiary and the renewal period is type 1, and F_2 is the fraction of time that the donor server helps the beneficiary and the renewal period is type 2. In fact, $F_2 = 0$. Therefore,

$$F = F_1 \leq \frac{E[R]}{E[Y]}.$$

This proves the necessity for $N_B^{th} > 0$. $\square$

Note that the right hand side of the stability condition for beneficiaries is an increasing function of N_D^{th}; in terms of stability, the larger N_D^{th}, the better the performance of beneficiaries. 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 when K_{sw} is exponentially distributed. 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, unless N_D^{th} is large. 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 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 $N_D^{th} = 10$.

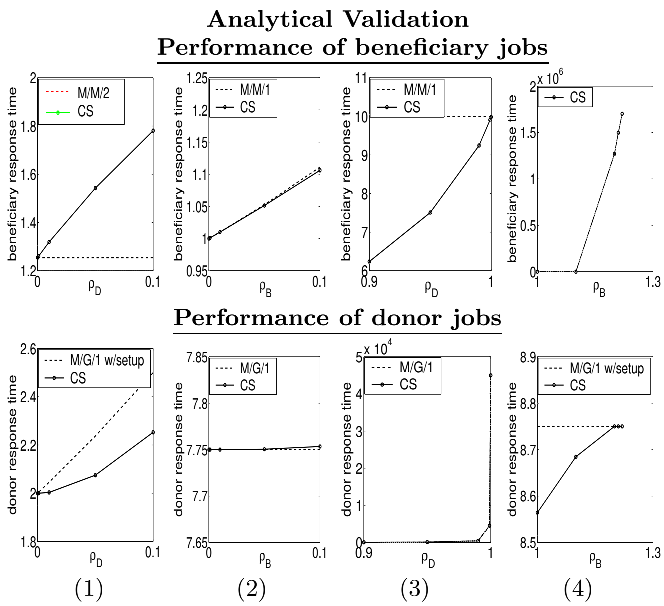


Figure 4: Validation of our cycle stealing analysis against four limiting cases. For lack of space we show graphs only for the specific parameters: $N_B^{th} = 3$, $E[K_{sw}] = E[K_{ba}] = 1$, X_B is exponentially-distributed with mean 1 and X_D is a Coxian with mean 1 and $C_D^2 = 8$. (1) $\rho_B = 0.9, \rho_D \to 0$; (2) $\rho_B \to 0, \rho_D = 0.6$; (3) $\rho_B = 0.9, \rho_D \to 1$; (4) $\rho_B \to 1.23, \rho_D = 0.6$.

5. VALIDATION OF ANALYSIS

Since our analysis involves approximation of busy periods by 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 four 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. We assume that X_D is generally distributed and X_B is exponentially distributed. For these limiting cases, the response times of the beneficiaries and donors are easy to evaluate since they converge in performance to either an M/GI/1, an M/GI/1 with setup cost, or an M/M/2.

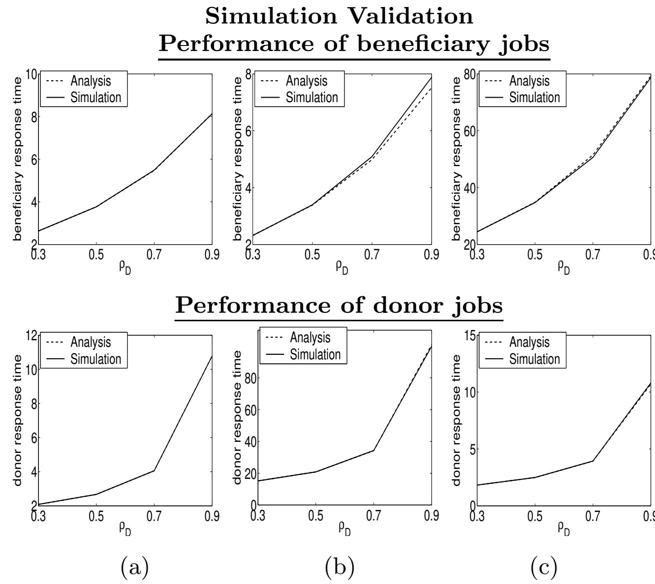


Figure 5: Validation of analysis against simulation. $E[K_{sw}] = E[K_{ba}] = 1$. We fix $\rho_S = 0.9$, and vary ρ_L. $N_B^{th} = 3$. X_B and X_D are exponentially-distributed. (a) $E[X_B] = 1, E[X_D] = 1$. (b) $E[X_B] = 1, E[X_D] = 10$. (c) $E[X_B] = 10, E[X_D] = 1$.

Figure 4 verifies that our analysis of cycle stealing has the correct limiting behavior in all of these cases. (Although we tested many different parameter settings, we show only a representative sample.) In case (1), $\rho_B = 0.9$ and $\rho_D \to 0$. The response time for beneficiary jobs converges to that under an M/M/2 queue, since the donor server can almost always help when $\rho_D \sim 0$. The response time for donor jobs converges to that under an M/GI/1 queue with setup cost K_{ba}. Since the interarrival time of donor jobs is huge when $\rho_D \sim 0$, the donor server is almost always at the beneficiary queue when a new donor job arrives, requiring a setup cost corresponding to the switching cost K_{ba}. In case (2), $\rho_D = 0.6$ and $\rho_B \to 0$. The response time for beneficiary jobs converges to that under an M/M/1 queue, since the number of beneficiary jobs is almost always $< N_B^{th}$ when $\rho_B \sim 0$ and hence there is no help from the donor server. The response time for donor jobs converges to that under an M/GI/1 queue, since the donor server is almost always at the donor queue and hence requires no setup cost. In case (3), $\rho_B = 0.9$ and $\rho_D \to 1$. The response time for beneficiary jobs converges to that under an M/M/1 queue, and the response time for donor jobs converges to that under an M/GI/1 queue, since the donor server is almost always busy working on donor jobs when $\rho_D \sim 1$ and hence there is no help from the donor server for the beneficiary and no setup cost for the donor. In case (4), $\rho_D = 0.6$ and $\rho_B \to \rho_B^{max}$. The response time for beneficiary jobs diverges to infinity as $\rho_B \to \rho_B^{max}$, as is predicted by the stability condition. The performance of donor converges to that under an M/GI/1 with setup cost K_{ba}, since the donor server almost always goes to help the beneficiary when it becomes idle and hence switching cost is almost always required to start working on a new donor job when $\rho_B \sim \rho_B^{max}$.

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 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. Throughout we will use the term "gain" to denote the improvement (drop) in mean response time experienced by beneficiary jobs under cycle stealing, as compared with dedicated servers. We will use the term "pain" to refer to the increase in mean response time experienced by donor jobs under cycle stealing as compared with dedicated servers. We define:

$$gain = \frac{E[T_B]^{\texttt{Dedicated}}}{E[T_B]^{\texttt{CS}}} \quad \text{and} \quad pain = \frac{E[T_D]^{\texttt{CS}}}{E[T_D]^{\texttt{Dedicated}}},$$

where $E[T_B]^{\texttt{Dedicated}}$ refers to the mean response time of beneficiaries under dedicated servers and $E[T_B]^{\texttt{CS}}$ refers to the mean response time of beneficiaries under cycle stealing; $E[T_D]^{\texttt{Dedicated}}$ and $E[T_D]^{\texttt{CS}}$ are defined similarly. Observe that both pain and gain have been defined to range from 1 to ∞, where infinite gain corresponds to the situation where the response time under dedicated is infinite and the response time under cycle stealing is finite.

6.1 Take-home messages

(Section 6.2) Cycle stealing is always a win when $\rho_{\mathbf{B}} > 1$, but doesn't pay when $\rho_{\mathbf{B}} < 0.5$. When $\rho_B > 1$ (and $\rho_D < 1$), cycle stealing can provide infinite gain to beneficiaries over dedicated servers, with comparatively little pain for the donors. This is because the *stability region* for the beneficiaries under cycle stealing is much greater than under dedicated servers. While factors such as increased switching costs, increased ρ_D, and increased N_B^{th} do lower the gain of the beneficiary, 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: $0.5 < \rho_B < 1$.

(Section 6.3) For $.5 < \rho_{\mathbf{B}} < 1$, cycle stealing has regions of high gain and 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 and also look at the *overall* mean response time (averaged over both beneficiary and donor jobs) to determine whether cycle stealing is "good" or "bad" overall. In general under higher ρ_B and lower ρ_D, cycle stealing is "good" overall, because the gain of the beneficiaries is so high in this region. 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. More important than the absolute switching cost is the switching cost relative to the mean donor job size. 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 find that while increasing N_B^{th} hurts the beneficiaries and helps the donors, increasing N_B^{th} (up to a point) nevertheless helps the overall mean performance, thus increasing the size of the "good" region.

(Section 6.4) For $.5 < \rho_{\mathbf{B}} < 1$, 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. We provide intuition for this surprising result. This echoes findings in [2] and [1] for a different model.

(Section 6.5) Setting the best threshold $\mathbf{N_B^{th}}$ is non-trivial. Finally we tackle the practical problem of how to set N_B^{th} in the region $.5 < \rho_B < 1$ so as to improve overall mean performance (averaged over both beneficiary and donor jobs). The precise effect of N_B^{th} on overall mean performance is irregular and delicate. We find that the optimal value of N_B^{th} seems to increase with ρ_D, decrease with ρ_B, and increase with switching cost.

Due to limited space we typically only show a couple of options for each parameter. Many more graphs are available in the associated technical report [10].

6.2 Benefits of cycle stealing: wide range $\rho_{\mathbf{B}}$

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 $\rho_B \geq 1$, cycle stealing can provide an unbounded gain to beneficiary jobs over dedicated servers, due to an increase in the stability region. 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 unbounded these factors are less important when $\rho_B \geq 1$. 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_{ba} This bound is tight for all N_B^{th} values as ρ_B reaches its maximum. Below this maximal stable load, 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 $\rho_B \leq 0.5$, 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: $.5 < \rho_{\mathbf{B}} < 1.0$

Figure 7 evaluates cycle stealing in the region $0.5 < \rho_B < 1.0$, as a function of ρ_D. In rows 1 and 2, X_B and X_D both have mean 1, while rows 3 and 4 assume X_B has mean 1 and

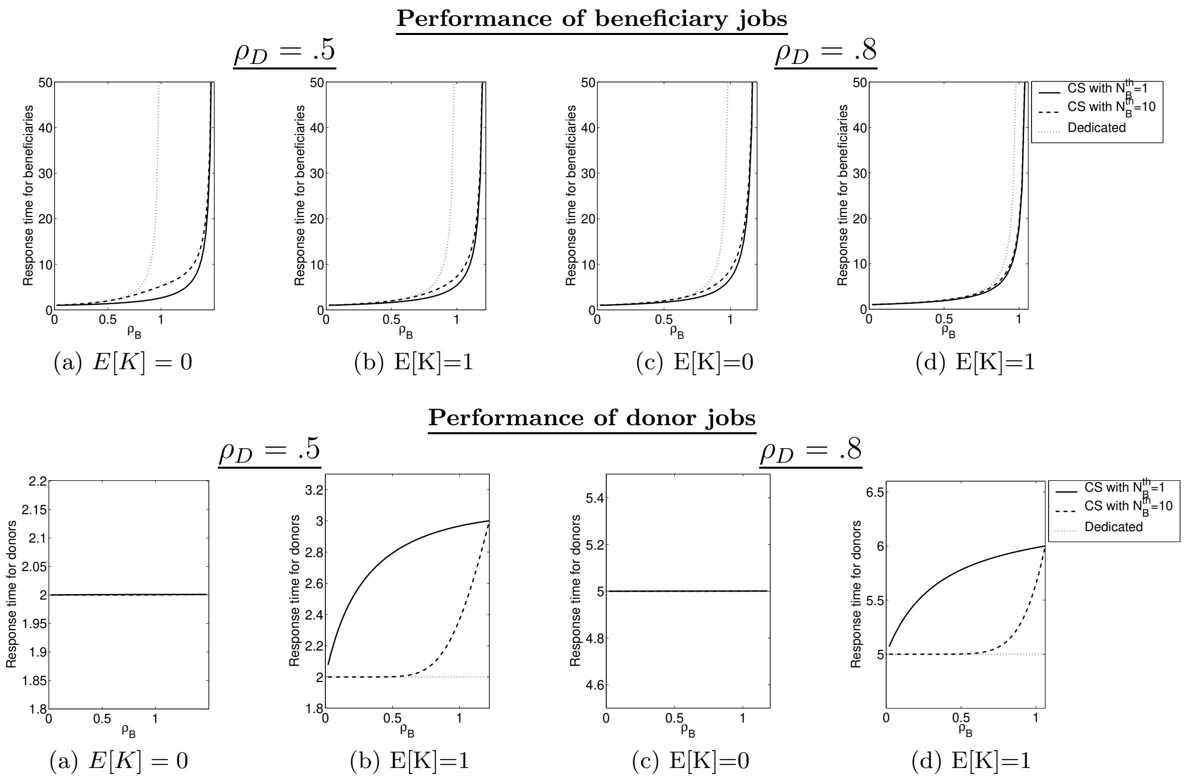


Figure 6: The response time for beneficiaries and donors as a function of ρ_B. Graphs show the case of (i) cycle stealing with $N_B^{th} = 1$, (ii) cycle stealing with $N_B^{th} = 10$, and (iii) dedicated servers. In all figures X_B and X_D are exponential with mean 1. Switching costs are exponential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

X_D has mean 10. Switching cost is 0 or 1, and N_B^{th} is 1 or 3, as labeled. In our discussion below, we first concentrate on the effect of switching cost, and then look at the effect of N_B^{th}.

Odd-numbered columns show various regions of beneficiary gain and donor pain. We define *low gain* as a beneficiary gain of 1.1 or less; *mid gain* as a beneficiary gain of between 1.1 and 1.5; and *high gain* as a beneficiary gain of over 1.5. Pain regions are defined similarly for donors. Even-numbered columns use the terms *good* and *bad*, where *good* indicates that the overall mean response time has dropped as a consequence of cycle stealing, and *bad* indicates the reverse.

Consider first rows 1 and 2 in Figure 7. The first and second columns show the case of zero switching cost. We see that all regions are low pain regions (in fact zero pain), 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 only slightly reduced gain for the beneficiaries. However, switching cost creates pain for the donor jobs. When ρ_D is very low, the pain appears high, but this is primarily due to the fact that "pain" is relative to the response time under dedicated servers, and the response time under dedicated servers is obviously low for small ρ_D. The overall mean benefit (over all jobs) is no longer always positive as shown in the fourth column of rows 1 and 2 in 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 pain for donors.

Rows 3 and 4 in Figure 7 differ from rows 1 and 2 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.

Focusing on columns 2 and 4 of Figure 7, which depict the effect on overall mean response time, we see that 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.

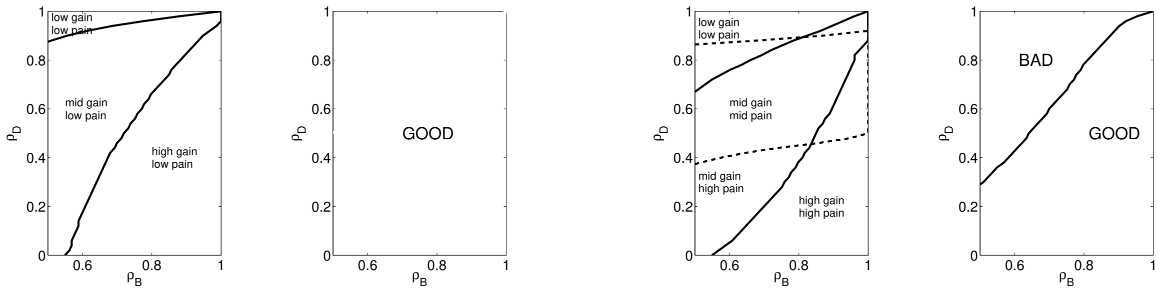
Finally, we discuss the effect of N_B^{th} in both figures. The performance of the beneficiaries is relatively insensitive to increasing N_B^{th} from 1 to 3 in both figures; however the donors benefit quite a bit from raising N_B^{th}. We further investigate the selection of N_B^{th} in Section 6.5.

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:

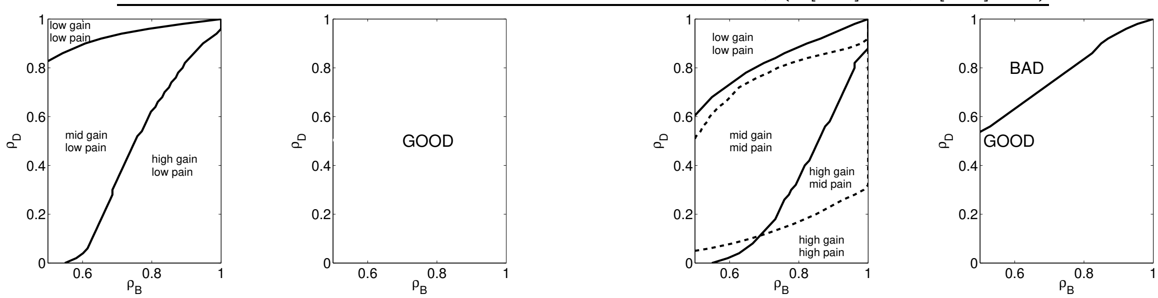
1. The donor pain should drop. This is because the donor response times will be higher overall and so the relative pain will appear diminished, and
2. The beneficiary gain should drop. This is because high

Gain of Beneficiaries and Pain of Donors — $N_B^{th} = 1$, $(E[X_B] = 1, E[X_D] = 1)$



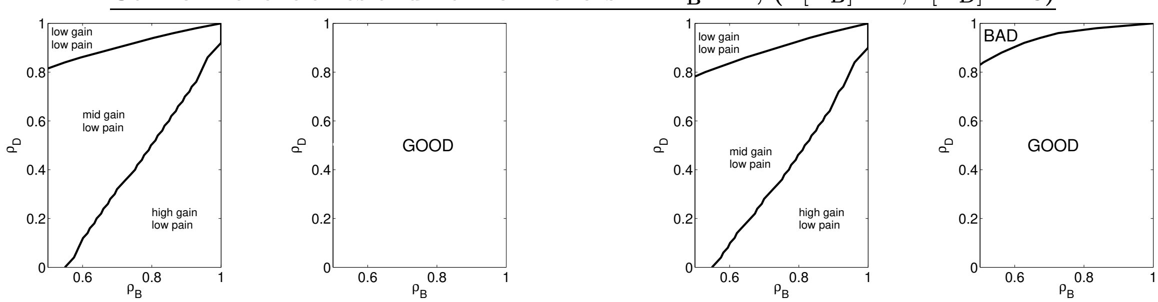
(a) $E[K] = 0$ (b) E[K]=1

Gain of Beneficiaries and Pain of Donors — $N_B^{th} = 3$, $(E[X_B] = 1, E[X_D] = 1)$



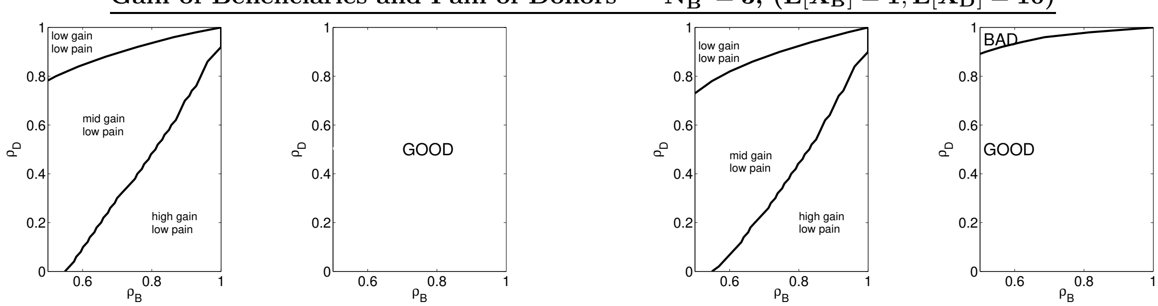
(a) $E[K] = 0$ (b) E[K]=1

Gain of Beneficiaries and Pain of Donors — $N_B^{th} = 1$, $(E[X_B] = 1, E[X_D] = 10)$



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

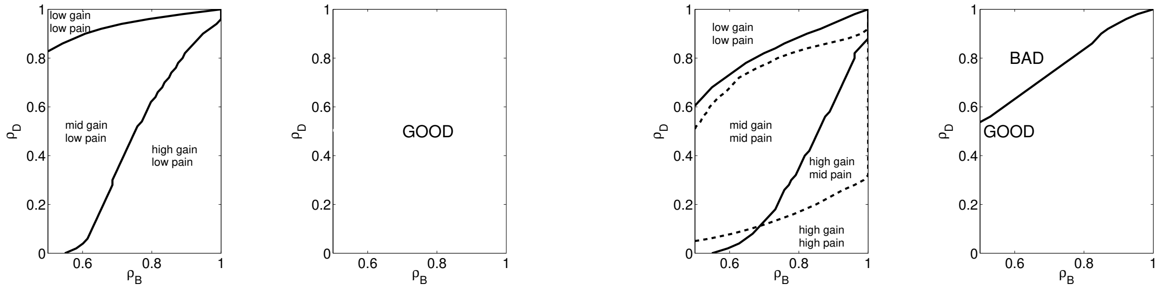
Gain of Beneficiaries and Pain of Donors — $N_B^{th} = 3$, $(E[X_B] = 1, E[X_D] = 10)$



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

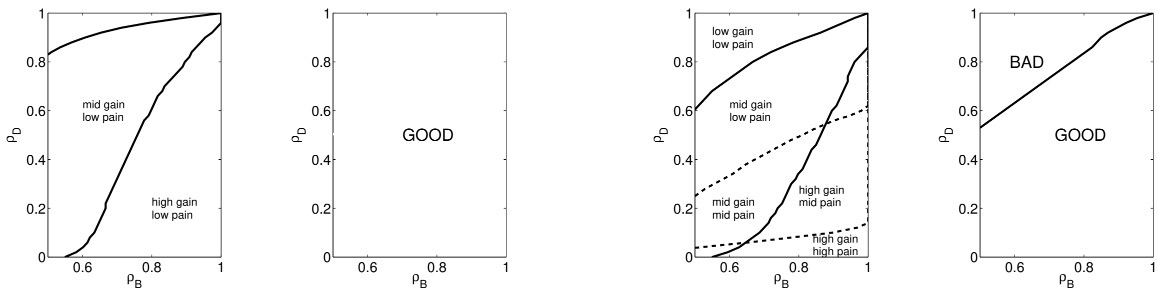
Figure 7: Columns 1 and 3 show the gain of beneficiary jobs and pain of donor jobs relative to dedicated servers. Solid lines delineate high/mid/low gain regions, and dashed lines delineate high/mid/low pain regions. Columns 2 and 4 show the effect of cycle stealing on the overall mean response time (over all jobs) relative to dedicated servers. In all figures, X_B and X_D are exponentially-distributed, where X_B and X_D both have mean 1 in rows 1 and 2, and X_B has mean 1 and X_D has mean 10 in rows 3 and 4.

Gain of Beneficiaries and Pain of Donors — $\mathbf{X_D}$ with $\mathbf{C_D^2 = 1}$



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

Gain of Beneficiaries and Pain of Donors — $\mathbf{X_D}$ with $\mathbf{C_D^2 = 8}$



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

Figure 8: 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, $N_B^{th} = 3$.

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 8 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: $C_D^2 = 1$) with row 2 (X_D has high variability: $C_D^2 = 8$), we see that there is no discernible difference in beneficiary performance.

To study this effect more closely, we next increase the variability in donor job sizes further. Figure 9 shows the performance of the beneficiary jobs under the case of zero switching cost, when the squared coefficient of variation of donor job size is 1, 8, or 50, and ρ_D is fixed at 0.5. We vary ρ_B from 0 to the stability condition. As is observed in Figure 8, the effect of variability of X_D on the performance of X_B is small when $\rho_B < 1$, and negligible when $\rho_B < 0.75$. When $\rho_B > 1$ the effect of variability in donor sizes may be significant. A critical factor seems to be whether the beneficiary queue is stable in isolation; when this is not the case high variability in donor visits leads to prolonged intervals of instability, which inflates the mean response time.

6.5 Setting the beneficiary threshold $\mathbf{N_B^{th}}$

This section seeks to determine a good value for the threshold N_B^{th}, as a function of the system parameters. We start with some obvious characteristics of N_B^{th}: (i) Increasing N_B^{th} leads to lower gain for the beneficiaries and lower pain for the donors. Perhaps less obvious, the relative drop in gain for the beneficiaries is slight compared to the drop in pain for the donors. This points towards choosing a higher value of N_B^{th}. (ii) If the switching cost is zero, the optimal N_B^{th} is 1 (or 0), since there is never any pain for the donors.

Figure 10 shows optimal values of N_B^{th} for minimizing overall mean response time (over all jobs) 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 optimal N_B^{th} at each load. For clarity we only show lines up to $N_B^{th} = 14$. The following additional characteristics of N_B^{th} 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}.

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 [10]. For example, in this paper we have assumed that the donor immediately switches back when a donor job arrives. 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 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

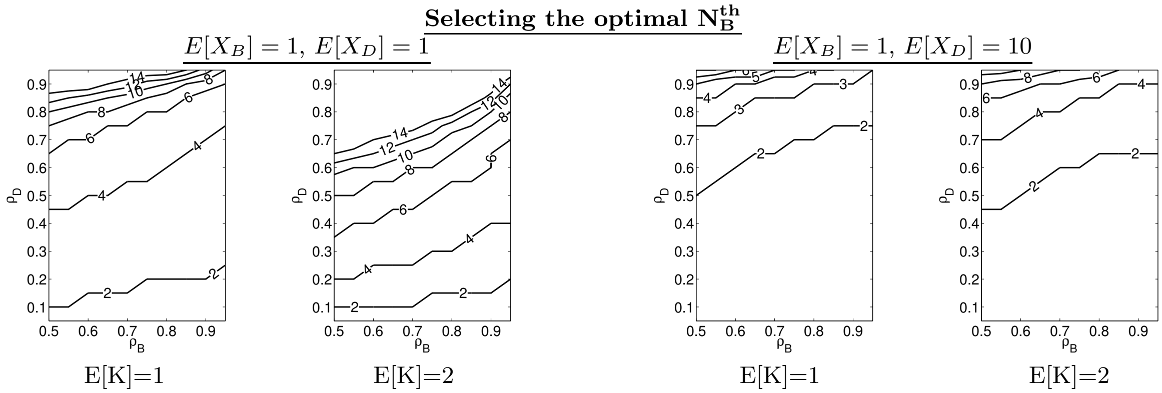


Figure 10: Graphs showing the optimal value of N_B^{th} with respect to overall mean response time.

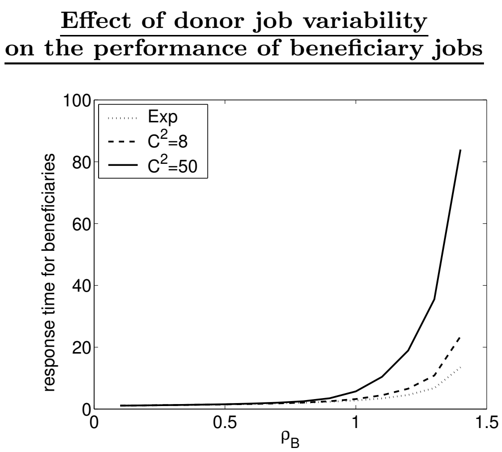


Figure 9: The response time for beneficiary jobs under different donor job size variability. We fix ρ_D at 0.5, and ρ_B varies from 0 to the stability condition of 1.5. The mean donor job size and beneficiary job size is 1, and switching cost is zero.

handle the problem of one beneficiary and two or more donor queues, 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. The case of multiple beneficiary queues does not seem readily solvable for the model in this paper but has been solved for a somewhat related model [5]. Another interesting question involves servers which function as both beneficiaries and donors. A model in which servers function as both beneficiaries and donors was looked at in [11]. Unlike our own model, in [11] there are assumed to be no dependencies between the servers. This means there is no need for a 2D-infinite chain, and no need for dimensionality reduction. In our, more complex model, the analysis of servers which function as both beneficiaries and donors is still open at the present time.

8. REFERENCES

[1] S. Borst, O. Boxma, and P. Jelenkovic. Reduced-load equivalence and induced burstiness in GPS queues with long-tailed traffic flows. *Queueing Systems*, (to appear).

[2] S. Borst, O. Boxma, and M. van Uitert. The asymptotic workload behavior of two coupled queues. *Queueing Systems*, 43:81–102, 2003.

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

[4] G. Fayolle and R. Iasnogorodski. Two coupled processors: the reduction to a Riemann-Hilbert problem. *Zeitschrift fur Wahrscheinlichkeitstheorie und vervandte Gebiete*, 47:325–351, 1979.

[5] M. Harchol-Balter, C. Li, T. Osogami, A. Sheller-Wolf, and M. S. Squillante. Cycle stealing under immediate dispatch task assignment. In *Proceedings of the Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, June 2003 (to appear).

[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. Necessary and sufficient conditions for representing general distributions by Coxians. Technical Report CMU-CS-02-178, School of Computer Science, Carnegie Mellon University, September 2002.

[10] T. Osogami, M. Harchol-Balter, and A. Sheller-Wolf. Analysis of cycle stealing with switching cost. Technical Report CMU-CS-02-192, School of Computer Science, Carnegie Mellon University, October 2002.

[11] M. S. Squillante and R. D. Nelson. Analysis of task migration in shared-memory multiprocessors. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 143–155, May 1991.

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