RC23093 (W0401-143) January 30, 2004
Mathematics

IBM Research Report

Cycle Stealing under Immediate Dispatch Task Assignment

Mor Harchol-Balter, Cuihong Li, Takayuki Osogami, Alan Scheller-Wolf
Carnegie Mellon University

Mark S. Squillante
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Cycle Stealing under Immediate Dispatch Task Assignment

Mor Harchol-Balter[*] Cuihong Li[†] Takayuki Osogami[‡] Alan Scheller-Wolf[§]
Mark S. Squillante[¶]

February 5, 2003

Abstract

We consider the practical problem of task assignment in a server farm, where each arriving job is immediately dispatched to a server in the farm. We look at the benefit of cycle stealing at the point of the dispatcher, where jobs normally destined for one machine may be routed to a different machine if it is idle. The analysis uses a technique which we refer to as dimensionality reduction via busy period transitions. Our analysis is approximate, but can be made as close to exact as desired, and is validated via simulation. Results show that the beneficiaries of the idle cycles can benefit unboundedly, while the donors are only slightly penalized. These results still hold even when there is only one donor server and 20 beneficiary servers stealing its idle cycles.

[*]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 01-1.

[†]Carnegie Mellon University, GSIA. cuihong@cs.cmu.edu

[‡]Carnegie Mellon University, CSD. osogami@cs.cmu.edu

[§]Carnegie Mellon University, GSIA. awolf@andrew.cmu.edu

[¶]IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598, mss@watson.ibm.com

1 Introduction

Server farm architecture

The "server farm" is a common architecture used by busy web sites, computation centers, file servers, and any other service which receives more requests than can be handled by a single server. The server farm is popular because it allows for increased computing power while being cost-effective and easily scalable. The server farm architecture is shown in Figure 1.

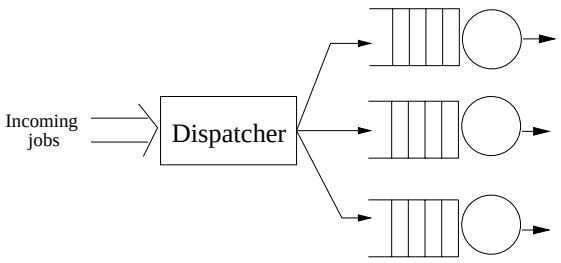


Figure 1: *Illustration of server farm architecture.*

In a server farm, each arriving "job" (request) is immediately dispatched by a high-speed front-end router (a.k.a., dispatcher) to exactly one of the servers, which handles the job. Common dispatchers include Cisco's Local Director [6] and IBM's Network Dispatcher [17]. The immediate dispatching of jobs is crucial for scalability and efficiency; it's important that the router not become a bottleneck. There is typically no communication between servers, which may not even know of each others' existence.

The rule used by the dispatcher for assigning jobs to servers is known as the *task assignment policy*. The choice of the task assignment policy has a significant effect on the performance perceived by users. Designing a distributed server system thus often comes down to choosing the "best" possible task assignment policy for the given model and user requirements. While devising new task assignment policies is easy, analyzing even the simplest policies can prove to be very difficult: Many of the long-standing open questions in distributed computing involve the performance analysis of task assignment policies.

In this paper we consider the *particular model* of a server farm in which servers are homogeneous and the execution of jobs is *non-preemptive* (run-to-completion), i.e., the execution of a job can't be interrupted and subsequently resumed. Our model is motivated by servers at supercomputing centers, where jobs are typically run-to-completion (see Table 1). Our model is also consistent with validated stochastic models used to study a wide range of high-volume Web sites [18, 34], studies of scalable systems for departmental computers within an organization [32], and telecommunication systems with heterogeneous servers [5].

Previous work on task assignment

The analysis of task assignment policies has been the topic of many papers. Below we provide a brief overview. We limit our discussion to task assignment in *non-preemptive* systems with *immediate dispatch*. For a more general discussion see [12] and the references therein.

By far the most common task assignment policy used is `Round-Robin`. The `Round-Robin` policy is simple, but it neither maximizes utilization of the servers, nor minimizes mean response time.

When the job sizes come from an exponential distribution, the best policy for minimizing mean response time is `Least-Remaining-Work`, where incoming jobs are sent to the server with the least total unfinished work [37]. This requires knowing the *size* of jobs (a.k.a., service requirements, processing requirements). In the case where the sizes are not known, then the `Shortest-Queue` task assignment policy – where incoming jobs are immediately dispatched to the server with the fewest number of jobs – has been shown to be optimal under an exponential job size distribution and Poisson arrival process [36, 10].

While policies like `Least-Remaining-Work` and `Shortest-Queue` perform well under *exponential* job size distributions, they perform *poorly* when the job size distribution has higher variability. In such cases, it has been shown analytically and empirically that the `Dedicated` policy far outperforms these other policies with respect to minimizing mean response time [13, 33]. In the `Dedicated` policy, some servers may be designated as the "short servers" and others as the "long servers." Short jobs are always sent to the short servers and long jobs to the long servers. The `Dedicated` policy is also popular in practice (e.g. Cornell Theory Center) where different server machines have different duration limitations: 0–1/2 hour, 1/2 – 2 hours, 2 – 4 hours, 4 – 8 hours, etc., and users must specify an estimated required service requirement for each job [16]. The intuition behind the `Dedicated` policy is

Name	Location	# Servers	Server Machine
Xolas [23]	MIT Lab for Computer Science	8	8-processor Ultra HPC 5000 SMP
Pleiades [22]	MIT Lab for Computer Science	7	4-processor Alpha 21164 machine
J90 distributed server	NASA Ames Research Lab	4	8-processor Cray J90 machine
J90 distributed server [1]	Pittsburgh Supercomputing Center	2	8-processor Cray J90 machine
C90 distributed server [2]	NASA Ames Research Lab	2	16-processor Cray C90 machine

Table 1: *Examples of server farms described by the architectural model of this paper. Observe that each server machine is a multi-processor machine. The schedulers used are Load-Leveler, LSF, PBS, or NQS. These schedulers typically only support run-to-completion (non-preemptive) within a server machine because (i) timesharing within a multiprocessor server is often not supported [29], and (ii) the huge memory requirements of these parallel jobs make timesharing too expensive [11]. Note: In such settings it is common for users to submit an upper bound on their job's CPU requirement in seconds; the job is killed if it exceeds this estimate.*

that, under high-variability workloads, it is important to isolate short jobs from the long jobs, as short jobs waiting behind long jobs is very wasteful. The `Dedicated` policy is also popular in supermarkets and banks, where a separate queue is created for "short" jobs.

Even when the job size is not known, it has been demonstrated that a policy very similar to `Dedicated`, known as the `TAGS` policy (Task Assignment by Guessing Size) works almost as well when job sizes have high variability. Like `Dedicated`, the `TAGS` policy significantly outperforms other policies that do not segregate jobs by size [12].

Motivation for cycle stealing

Given the extremely high variability of job sizes under so many computer workloads [8, 9, 14, 33, 24, 31, 30], `Dedicated` assignment is clearly preferable to other policies. However `Dedicated` is still clearly *not* optimal. One problem is that `Dedicated` can lead to situations where the servers are not fully utilized: five consecutive short jobs may arrive, with no long job, resulting in an idle long server. This is especially likely in common computer workloads, where there are many short jobs and just a few very long jobs, resulting in longer idle periods between the arrivals of long jobs.

Ideally one would like a policy which combines the variance-reducing benefit of the `Dedicated` policy with the high-utilization property of other policies: We would like to segregate jobs by size so as to provide isolation for short jobs, but during times when the long job server is free, we would like to *steal* the long server's idle cycles and use those to serve incoming short jobs. This would both decrease the mean response time of short jobs, and also enlarge the *stability region* of the overall system. It is important, though, that we permit the short jobs to use the long server only when that server is *free*, so that we don't *starve* the long jobs. Nonetheless, because jobs are not preemptible, there will still be some penalty to a long job which arrives to find a short job serving at the long server. Our specific cycle stealing algorithm, called `CS-Immediate-Dispatch`, will be described in Section 2.

Beneficiaries and donors

Above we've used the terms "short server" to describe the server designated for "short" jobs and "long server" to describe the server designated for "long" jobs, but which can be used for new short arrivals if idle. Our reason for talking about a "short server" and a "long server" is to emphasize the tremendous performance benefit achievable when jobs can be segmented by size. The analysis in this paper, however, applies more generally to any situation where there is a "beneficiary server" and a "donor server", where newly arriving beneficiary jobs may use the donor server if it is idle. Throughout, we will therefore use the terms *beneficiary* and *donor* jobs/servers. For completeness we will consider three cases: beneficiary jobs shorter than donor jobs; beneficiary jobs indistinguishable from donor jobs; and beneficiary jobs longer than donor jobs. We will find that the beneficiary jobs benefit in all three cases. The donor jobs suffer little, except in the case where the beneficiary jobs are much longer than donor jobs, causing donor jobs to sometimes get stuck waiting. But even in this case, it will turn out that the penalty to the donor jobs is dominated by the benefits to the beneficiary jobs.

Difficulty of analysis and new analytic approaches

Cycle stealing is a very old concept, and policies based on cycle stealing have been suggested in countless pa-

pers. However until this year ([15, 28]) the analysis of cycle stealing has eluded researchers. This paper provides the first analysis of cycle stealing under immediate dispatch task assignment. Our primary goal is to derive the *mean response time* for the beneficiaries and the *mean response time* for the donors.* Observe that even for the simplest instance of our problem – where job arrivals are Poisson and beneficiary jobs and donor jobs are drawn i.i.d. from respective exponential distributions – the continuous-time Markov chain representation of the system is mathematically intractable. This is due to the fact that the state space

$$(\textit{number beneficiary jobs}, \textit{number donor jobs})$$

grows infinitely in *two* dimensions (2D), without any simplifying structure. While truncation of the Markov chain is possible, the errors introduced by ignoring portions of the state space (infinite in 2D) can be quite significant, especially at higher traffic intensities†. Thus truncation is not sufficiently accurate nor robust for our purposes.

Our approach consists of several ideas. First, we define our `CS-Immediate-Dispatch` algorithm in a way that will allow the *decomposition* of the system into two processes: the beneficiary server process and the donor server process. We can solve the donor server process exactly, providing a closed-form expression for the Laplace transform of response time of donor jobs. We can also derive all moments of the busy period duration for the donor server, where the donor server's busy period is defined as the time from when the server becomes busy until it becomes idle again. We next analyze the beneficiary server. Normally this would require tracking both the number of donor and beneficiary jobs in a 2D-infinite (intractable) chain. However, we show that we can extract all the information we need by tracking only the number of beneficiary jobs and whether the donor server is busy or not. To do this we use a special type of transition in our Markov chain which we call *a busy period transition*, and which represents the length of the donor server's busy period. These transitions allow us to represent the beneficiary server state using a 1D infinite chain. This chain can be easily solved using known numerical (Matrix analytic) techniques. While a closed-form solution is 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 curves in this paper.

The only approximation in our method lies in the accuracy of the representation of the donor server's busy period. This can be made as accurate as desired, as all moments of the donor server's busy period are known. In this paper we match 3 moments and verify via simulation that this is sufficient. To summarize, we are able to analyze this heretofore intractable problem by (i) defining our policy so as to allow decomposition of the server states, and (ii) using busy period transitions to reduce dimensionality.

The above analysis allows for *very general conditions*. The service requirements of the beneficiary and donor jobs are assumed to be drawn i.i.d. from any general distribution (which we model using a Coxian distribution [7]).‡ The arrival process is assumed to be Poisson, but can easily be generalized to a MAP process (Markovian Arrival Process) [27]. The analysis also generalizes nicely to the case of n beneficiary servers.

The work in this paper complements two other papers, not yet appeared: [15] and [28]. These other papers also look at the question of cycle stealing, however under different models which are not appropriate for server farms.

In [15], there is a central queue at the dispatcher and no queueing at the servers. This model lacks practicality for server farms, but is interesting theoretically since it leads to a much larger stability region as compared to immediate dispatch, since donors can help with *all* beneficiary jobs, not just new arrivals. The analysis of the central queue policy does not lend itself to decomposition of the servers (as in this paper). Therefore the authors in [15] are forced to make certain independence approximations in analyzing their model.

In [28], the model differs still further. Similarly to [15], the donors can help with all beneficiary jobs, not just new arrivals. In addition, the service is preemptive, which means that the donor server quits work on a beneficiary job when new donors arrive. The preemp-

*The response time is defined as the time from when the job arrives until it leaves the system.

†For $\rho_B = 1.1$ and $\rho_D = 0.7$, truncation leads to $\sim 25\%$ error, even with 2×50^2 states, and takes > 10 minutes to compute (here ρ_B represents the load at the beneficiary server and ρ_D the load at the donor server). As ρ_B nears 1.16119 ($\rho_B < 1.16119$ is necessary for stability when $\rho_D = 0.7$) , the error increases indefinitely. Under job sizes more variable than exponential, the error is greater.

‡Coxian distributions are a subset of phase-type distributions. Phase-type distributions are commonly used for representing general distributions as a combination of exponential distributions of different rates, which allows the general distribution to be modeled in a Markov chain.

tive service model greatly simplifies the analysis, which allows the authors in [28] to analyze additional effects like switching costs and thresholds as well.

In both [28] and [15] it does not appear possible to analyze the case of multiple beneficiary servers, as in this paper.

Outline

Section 2 presents the `CS-Immediate-Dispatch` algorithm. Sections 3 through 5 discuss the case of a single beneficiary server and a single donor server, including: the analysis of `CS-Immediate-Dispatch` (Section 3); stability criteria (Section 4); and results showing the improvement of cycle stealing for beneficiary jobs and the penalty to donor jobs (Section 5) as a function of load, job sizes, and variability in the job size distributions. Section 6 discusses analysis and results for the case of multiple beneficiary servers. In Section 7 we validate our analysis technique against simulation and limiting cases.

Summary of Results

Our analysis leads to the following conclusions:

We find that immediate dispatch with cycle stealing vastly improves the performance of the beneficiaries. By increasing the stability region for the system, `CS-Immediate-Dispatch` results in finite response time even when the beneficiary load exceeds 1. We find that the donor jobs do not suffer much by having their idle cycles stolen. When the beneficiary jobs are smaller on average than the donors, the suffering of the donors is truly negligible. Even when the beneficiary jobs are ten times larger on average than the donors, the donor jobs' response times increase by only a small factor, and this increase is dwarfed by the "infinite" improvement possible for beneficiary jobs.

The performance of the beneficiary jobs is surprisingly insensitive to the variability of the donor job size distribution, with noticeable impact only in the region where the beneficiary load exceeds 1. The performance of donor jobs is negatively impacted by higher variability in beneficiary jobs.

We find, importantly, that the benefit to beneficiary jobs remains substantial and the penalty to donor jobs remains relatively low, *even when there are 20 beneficiary servers*. This underscores the power of cycle stealing and is important news for many applications outside of server farms, e.g., the sharing of unused bandwidth by multiple flows in Weighted Fair Queueing (WFQ) routers. Interestingly, we see that the change in going from i to $i+1$ beneficiary servers is strongly nonlinear. There is a big drop-off in going from 1 to 2 beneficiary servers, but a much smaller change in going from 5 to 20 beneficiary servers.

2 Algorithm and notation

The natural immediate-dispatch cycle stealing algorithm might look something like this: Donor jobs are immediately dispatched to the donor server. Beneficiary jobs are immediately dispatched to the beneficiary server, unless the beneficiary server is busy and the donor server is idle, in which case the beneficiary job goes to the donor server. Unfortunately, this algorithm does not appear tractable because the stochastic process defining the system cannot be decomposed: Even the analysis of only the donor server requires keeping track of both the number of beneficiary jobs and the number of donor jobs, which is still a 2D-infinite chain.

Fortunately, if we perturb the algorithm just slightly, we produce a policy which we can decompose and analyze using the method of dimensionality reduction via busy period transitions. Our algorithm is called `CS-Immediate-Dispatch` (cycle stealing under immediate dispatch). There is a designated beneficiary job server and a designated donor job server. An arriving donor job is always dispatched to the donor job server. An arriving beneficiary job first checks if the *donor* job server is idle. If so, the beneficiary job is dispatched to the donor job server (regardless of the number of jobs at the beneficiary queue). If however the donor job server is not idle (either it's working on a donor job or a beneficiary job), then the arriving beneficiary job is dispatched to the beneficiary server. Jobs at a server are serviced in FCFS order. The `CS-Immediate-Dispatch` algorithm is shown in Figure 2.

The `CS-Immediate-Dispatch` algorithm is an improvement over simple `Dedicated`, since a fraction of the arrival stream of beneficiary jobs can be offloaded to the donor server, while hardly penalizing donor jobs.

In the more general case (see Section 6), there may be several beneficiary servers and a single donor server. Jobs destined for beneficiary server i will first see check if the donor server is idle and if so go there. If not, they will go to their designated beneficiary server.

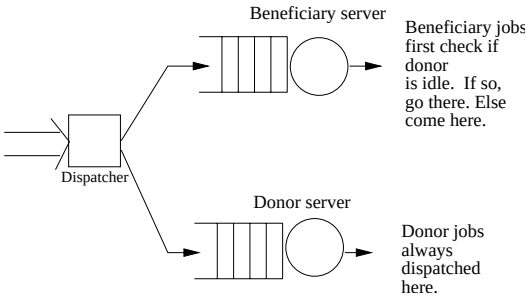


Figure 2: `CS-Immediate-Dispatch` *algorithm.*

Throughout we assume that beneficiary (respectively, donor) jobs arrive according to a Poisson process with rate λ_B (respectively, λ_D). The size, a.k.a. service requirement, of beneficiary jobs (respectively, donor jobs) is denoted by the random variable (r.v.) X_B (respectively, X_D) and is assumed to be represented by a Coxian distribution. The ith moment of the size of a beneficiary (respectively, donor) job is therefore $E[X_B^i]$ (respectively, $E[X_D^i]$). The load at the beneficiary server (respectively, donor server) is defined to be $\rho_B = \lambda_B \cdot E[X_B]$ (respectively, $\rho_D = \lambda_D \cdot E[X_D]$). In the case of multiple beneficiary servers, we have arrival rates: $\lambda_{B_1}, \lambda_{B_2}, \ldots, \lambda_{B_n}$, with job sizes $X_{B_1}, X_{B_2}, \ldots, X_{B_n}$, respectively. We assume that the first three moments of the busy periods are finite, and the queues are stable.

3 Analysis for two servers

Our analytic approach involves a three-step process: analyzing response time for the donor jobs; deriving the donor server busy period; and analyzing the beneficiary server by incorporating results from the donor server busy period.

The first step is accomplished by analyzing the donor job server. Observe that the state of the donor server is independent of the state of the beneficiary server: The donor server receives arrivals at rate $\lambda_B + \lambda_D$ when it is idle, but receives arrivals at rate only λ_D when it is busy. Furthermore, when the donor server is idle, the probability that the next arrival is a beneficiary job, as opposed to a donor job is $\frac{\lambda_B}{\lambda_B+\lambda_D}$. The simplest way to analyze such a system is via virtual waiting time analysis. To avoid disturbing the flow of the paper, we postpone the proof of this theorem to Appendix B.

Theorem 3.1 *Let $T^{(D)}$ represent the response time of donor jobs and let $\widetilde{T}^{(D)}(s)$ denote it's Laplace transform. Then:*

$$\widetilde{T}^{(D)}(s) = \frac{s+\lambda_B-\widetilde{X}_B(s)\lambda_B}{s-\lambda_D+\widetilde{X}_D(s)\lambda_D}\cdot\pi_0\cdot\widetilde{X_D}(s);$$

$$\mathsf{E}\left[T^{(D)}\right] = \frac{\rho_D}{1-\rho_D}\frac{\mathsf{E}\left[X_D^2\right]}{2\mathsf{E}\left[X_D\right]}+\frac{\rho_B}{1+\rho_B}\frac{\mathsf{E}\left[X_B^2\right]}{2\mathsf{E}\left[X_B\right]} + \mathsf{E}\left[X_D\right];$$

$$\mathsf{E}\left[(T^{(D)})^2\right] = \frac{\rho_D}{1-\rho_D}\frac{\mathsf{E}\left[X_D^3\right]}{3\mathsf{E}\left[X_D\right]}+\frac{\rho_B}{1+\rho_B}\frac{\mathsf{E}\left[X_B^3\right]}{3\mathsf{E}\left[X_B\right]} + \frac{\rho_D}{1-\rho_D}\frac{\mathsf{E}\left[X_D^2\right]}{\mathsf{E}\left[X_D\right]}(\mathsf{E}\left[T^{(D)}\right]-\mathsf{E}\left[X_D\right]) + 2\mathsf{E}\left[X_D\right]\mathsf{E}\left[T^{(D)}\right]+\mathsf{E}\left[X_D^2\right]-2\mathsf{E}\left[X_D\right]^2;$$

where

$$\pi_0 = \frac{1-\rho_D}{1+\rho_B}.$$

Here π_0 represents the fraction of time that the donor server is idle.

Our second step is to derive all moments for the busy and idle periods of the donor server, for use in our analysis of the beneficiary server. The donor server idle time is exponentially-distributed with rate $\lambda_D + \lambda_B$. To derive the length of the busy period for the donor server, we need to distinguish between two types of busy periods: (1) A busy period made up entirely of donor jobs, whose duration is represented by the r.v. B_D; and (2) A busy period started by one beneficiary job followed by zero or more donor jobs, whose duration is represented by the r.v. B_{BD}. We then have the following Laplace transforms:

$$\widetilde{B_D}(s) = \widetilde{X_D}(s+\lambda_D-\lambda_D\widetilde{B_D}(s));$$
$$\widetilde{B_{BD}}(s) = \widetilde{X_B}(s+\lambda_D-\lambda_D\widetilde{B_D}(s)).$$

From these transforms we compute the first three moments of each type of busy period as follows:

$$\mathsf{E}\left[B_D\right] = \frac{\mathsf{E}\left[X_D\right]}{1-\rho_D}; \qquad \mathsf{E}\left[B_{BD}\right] = \frac{\mathsf{E}\left[X_B\right]}{1-\rho_D};$$

$$\mathsf{E}\left[B_D^2\right] = \frac{\mathsf{E}\left[X_D^2\right]}{(1-\rho_D)^3};$$

$$\mathsf{E}\left[B_{BD}^2\right] = \mathsf{E}\left[X_B\right]\cdot\lambda_D\cdot\frac{\mathsf{E}\left[X_D^2\right]}{(1-\rho_D)^3}+\frac{1}{(1-\rho_D)^2}\cdot\mathsf{E}\left[X_B^2\right];$$

$$\mathsf{E}\left[B_D^3\right] = \frac{3\lambda_D(\mathsf{E}\left[X_D^2\right])^2}{(1-\rho_D)^5}+\frac{\mathsf{E}\left[X_D^3\right]}{(1-\rho_D)^4};$$

$$\mathsf{E}\left[B_{BD}^3\right] = \frac{\lambda_D^2\mathsf{E}\left[X_B\right]\cdot 3\cdot(\mathsf{E}\left[X_D^2\right])^2}{(1-\rho_D)^5}+\frac{\mathsf{E}\left[X_B^3\right]}{(1-\rho_D)^3} + \frac{3\lambda_D\mathsf{E}\left[X_D^2\right]+\mathsf{E}\left[X_B^2\right]+\lambda_D\mathsf{E}\left[X_B\right]\mathsf{E}\left[X_D^3\right]}{(1-\rho_D)^4}.$$

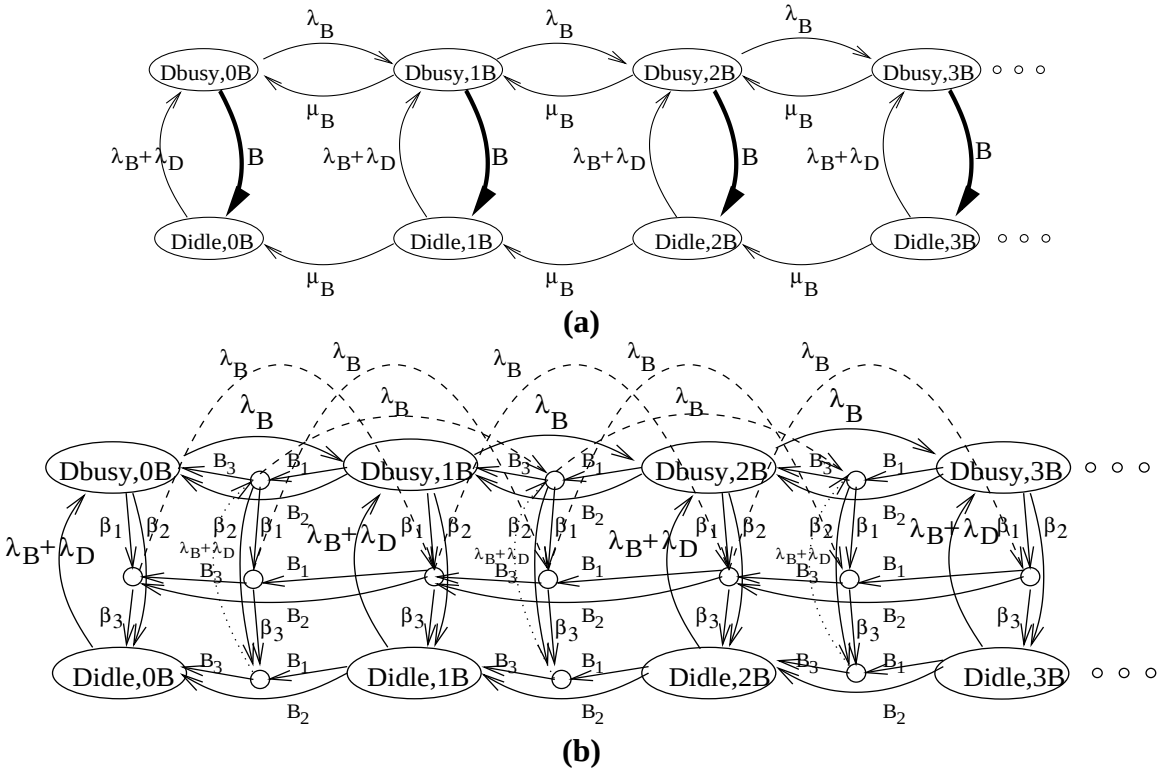


Figure 3: *Markov chain for beneficiary server. (a) Where job sizes are exponentially-distributed. (b) Where job sizes are Coxian.*

To obtain the moments of a general busy period for the donor server, observe that, due to Poisson arrivals, a busy period is of type B_{BD} with probability $\lambda_B/(\lambda_B + \lambda_D)$ and of type B_D with probability $\lambda_D/(\lambda_B + \lambda_D)$. Hence, denoting by B the duration of a general busy period, we have

$$\mathsf{E}\left[B^\ell\right] = \frac{\lambda_B}{\lambda_B + \lambda_D}\mathsf{E}\left[B_{BD}^\ell\right] + \frac{\lambda_D}{\lambda_B + \lambda_D}\mathsf{E}\left[B_D^\ell\right].$$

We now construct a phase-type distribution to match as many moments of the donor server's busy period as are of interest. Many methods exist for matching moments to phase-type distributions: [19, 4, 20]. We find that simply fitting a 2-stage Coxian distribution to the first three moments of the donor server's busy period works sufficiently well for our purposes.

Our last step is to analyze the beneficiary server. Analysis of the beneficiary server would seem to require a 2D-infinite Markov chain which tracks the number of donor and beneficiary jobs. However, if we use busy period transitions, a 1D-infinite chain suffices as follows: Observe that the arrival rate at the beneficiary server is λ_B during times when the donor server is busy and 0 during times when the donor server is idle. To represent the beneficiary server, we therefore only need to track the number of beneficiary jobs (1D-infinite), while maintaining a binary state recording whether the donor server is busy. Figure 3(a) shows the Markov chain model for the beneficiary server, under the simplification that job sizes are exponentially-distributed. Here the busy period duration for the donor server is shown as a single bold transition marked B. Figure 3(b) is the Markov chain that we actually solve for the beneficiary server. Here the job sizes are drawn from a 2-stage Coxian distribution, used to match the first 3 moments of the respective job size distribution. The length of the busy period for the donor server is also matched for the first 3 moments by a 2-stage Coxian.

We solve the Markov chain in Figure 3(b) for the number of beneficiary jobs at the beneficiary server using well-known Matrix Analytic methods[§]. Then via Little's Law[25], we obtain the mean response time of

[§]The Matrix Analytic method [26, 21] is a compact and fast method for solving QBD (quasi-birth-death) Markov chains which are infinite in only one dimension, where the chain repeats itself after some point. The repeating portion is represented as powers of a generator matrix which can be added as one adds a geometric series to produce a single matrix. Every curve in this paper which used Matrix Analytic analysis was produced within a couple seconds using the Matlab 6 environment.

small jobs which serve at the beneficiary server. Aggregating beneficiary jobs at both servers, we then have by Poisson-Arrivals-See-Time-Averages [37]:

$$\mathsf{E}\left[\text{ Time for beneficiary jobs }\right] =$$
$$\Pr\{\text{Donor server idle}\} \cdot \mathsf{E}\left[X_B\right] +$$
$$\Pr\{\text{Donor server busy}\} \cdot \mathsf{E}\left[\text{Time at benefic. server}\right].$$

Observe that while the mean response time for the donor jobs is exact, the mean response time for beneficiary jobs is an approximation which depends on the accuracy of the approximation of the busy period of the donor server. We have matched the first three moments of the busy period of the donor server. Greater accuracy can be achieved by matching more moments, by using a higher degree Coxian, until in theory the results are arbitrarily close to the actual quantities.

4 Stability conditions

For `Dedicated` assignment it is required that $\rho_D < 1$ and $\rho_B < 1$, where ρ_D (respectively, ρ_B) denote the load made up of donor jobs (respectively, beneficiary jobs). For `CS-Immediate-Dispatch` we will see that the region of stability is much wider. A proof of the following theorem is in Appendix A:

Theorem 4.1 *The stability condition for donor jobs is* $\rho_D < 1$*, and the stability condition for beneficiary jobs is the solution to:* $\rho_D < \frac{1}{\rho_B} + 1 - \rho_B$.

The restriction on ρ_B for `Dedicated` and `CS-Immediate-Dispatch` is shown in Figure 4. Observe the advantage of cycle stealing in extending the stability region. When ρ_D is near zero, ρ_B can be as high as 1.6.

5 Results of analysis for two servers

In this section we evaluate the results of our analysis. All figures are organized into two parts: the benefit to beneficiary jobs and the penalty to donor jobs. To evaluate these benefits/penalties we compare with the `Dedicated` algorithm which involves no cycle stealing. In all results figures, we hold ρ_D fixed and consider the full range of stable ρ_B for three sets of mean job sizes: beneficiaries have mean size 1 and donors have mean size 1; beneficiaries have mean size 1 and donors have mean size 10; beneficiaries have mean size 10 and donors have mean size 1.

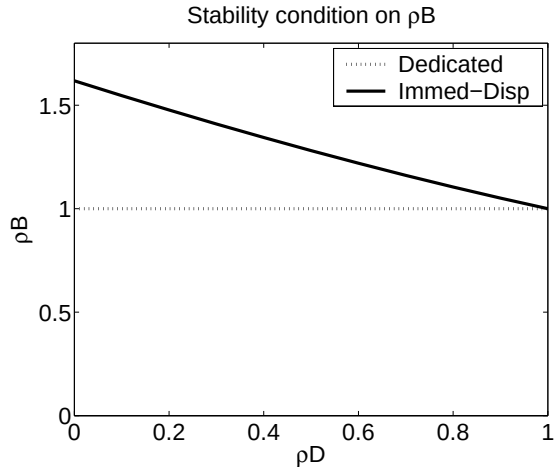


Figure 4: *Stability region on* ρ_B *for* `Dedicated` *and* `CS-Immediate-Dispatch`.

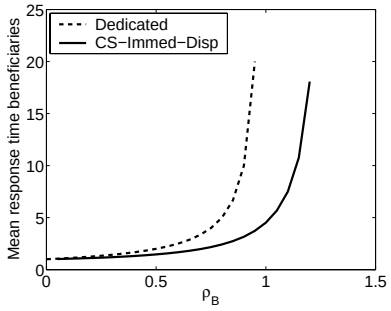
We generated results for various beneficiary and donor jobs size distributions. Due to space limitations, we show only a small subset of the result plots generated, but include the broader picture in our discussion below. In Figure 5 we assume that beneficiary job sizes are drawn from an exponential distribution and donor job sizes are drawn from a Coxian distribution with squared coefficient of variation, $C^2 = 8$.[¶] In Figure 6 we consider job size distributions with a range of C^2 values.

Looking at Figure 5, we see that the benefit to the *beneficiary jobs* is unbounded as $\rho_B \to 1$, since the mean response time for beneficiary jobs is *infinite* under `Dedicated`, and only a small *finite* value under `CS-Immediate-Dispatch` (specifically, 5 when $\rho_D = 0.5$, and 15 when $\rho_D = 0.8$, in column (a) where beneficiaries and donors have the same mean size). Note: graphs have been truncated so as to fit on the page.

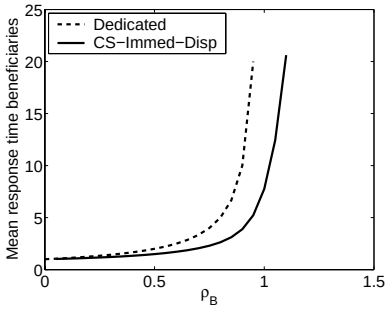
By comparison, the penalty imposed on *donor jobs* by cycle stealing is always relatively small in our experiments. This penalty increases with ρ_B, but not greatly. Looking at Figure 5(row 2), we see that even when $\rho_B = 1$, the penalty to donor jobs is only 10% for the case where beneficiaries and donors are equal and only 1% for the case where beneficiaries are shorter than donors. In the pathological case where beneficiaries are longer than donors, the penalty is greater. This is to be expected since jobs are not preemptible and a

[¶]The squared coefficient of variation, C^2, is defined to be the variance divided by the squared mean.

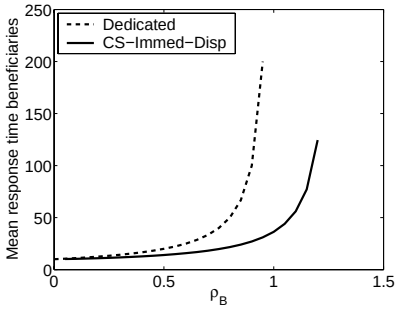
How beneficiaries gain from cycle-stealing – $\rho_D = 0.5$



(a) beneficiaries 1, donors 1

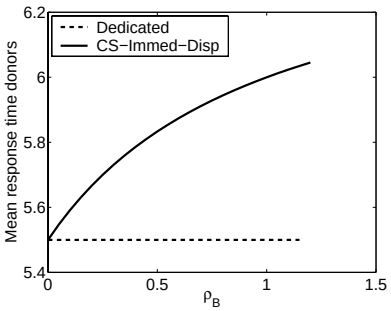


(b) beneficiaries 1, donors 10

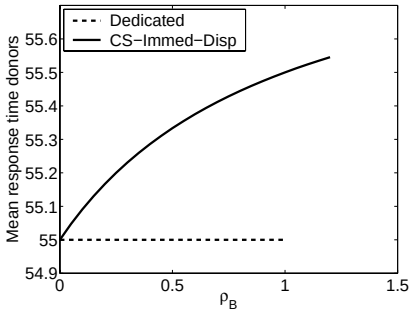


(c) beneficiaries 10, donors 1

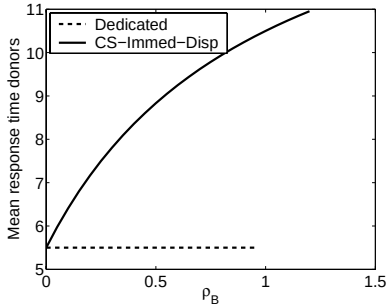
How donors suffer from cycle-stealing – $\rho_D = 0.5$



(a) beneficiaries 1, donors 1

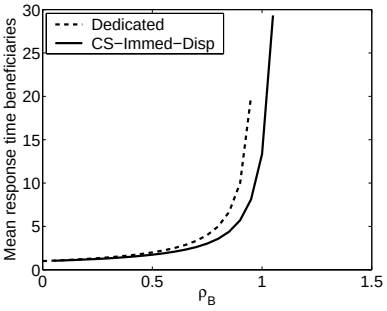


(b) beneficiaries 1, donors 10

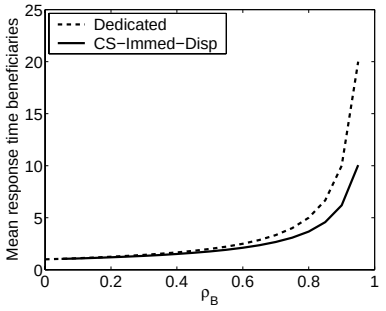


(c) beneficiaries 10, donors 1

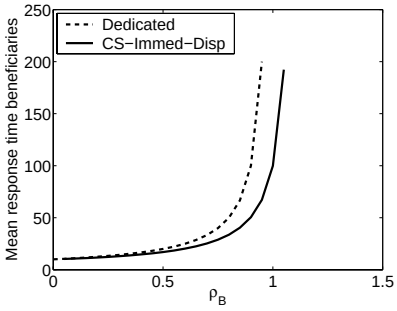
How beneficiaries gain from cycle-stealing – $\rho_D = 0.8$



(a) beneficiaries 1, donors 1

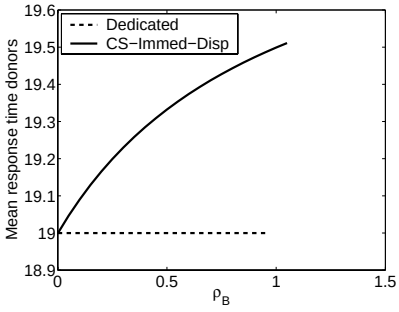


(b) beneficiaries 1, donors 10

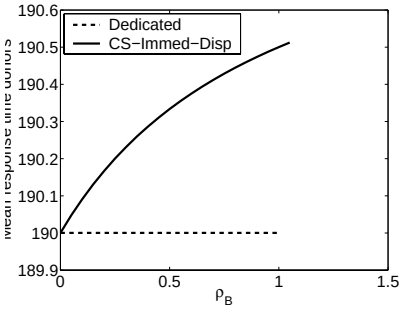


(c) beneficiaries 10, donors 1

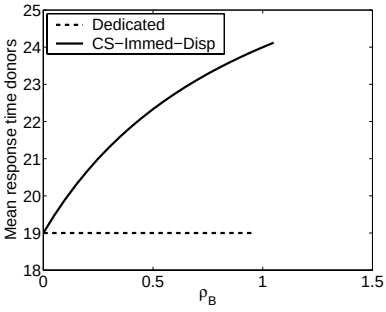
How donors suffer from cycle-stealing – $\rho_D = 0.8$



(a) beneficiaries 1, donors 1

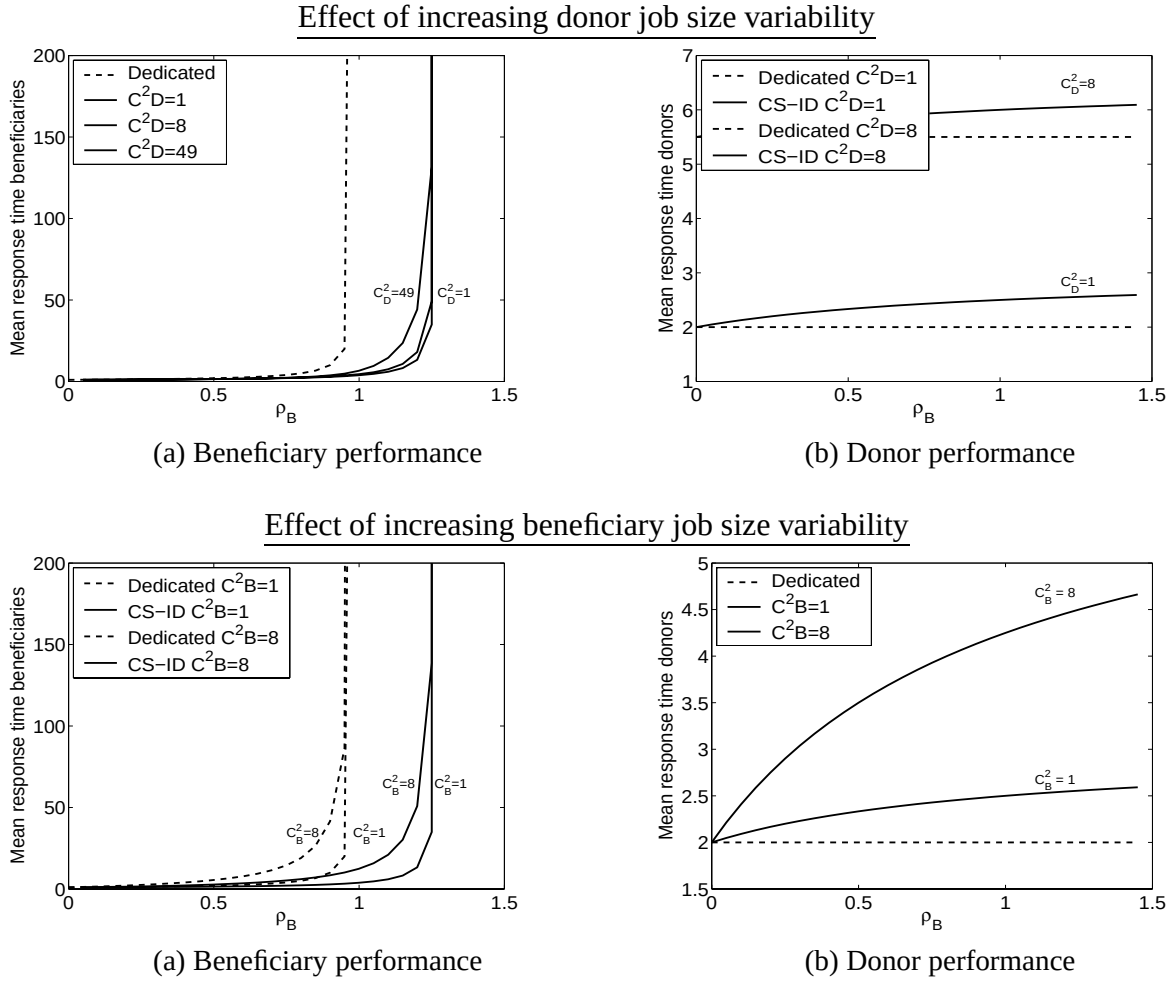


(b) beneficiaries 1, donors 10



(c) beneficiaries 10, donors 1

Figure 5: *Results of analysis for 2 servers, in the case where donors are drawn from Coxian distribution with appropriate mean and* $C^2 = 8$ *and beneficiaries are drawn from an exponential distribution with appropriate mean. (a)* $E[X_B] = 1$ *;* $E[X_D] = 1$*; (b)*$E[X_B] = 1$*;* $E[X_D] = 10$*; (c)* $E[X_B] = 10$*;* $E[X_D] = 1$*. Note different scales.*



(a) Beneficiary performance (b) Donor performance

(a) Beneficiary performance (b) Donor performance

Figure 6: *Effect of variability in donor job size and beneficiary job size on performance.*

donor job may now get stuck waiting behind a beneficiary job 10 times its size. Observe that the relative penalty to donor jobs is significantly reduced at higher donor loads ($\rho_D = 0.8$).

Figure 6 considers the effect of increasing the job size variability of donors and of beneficiaries. Increasing donor job size variability (from $C_D^2 = 1$ to $C_D^2 = 8$ to $C_D^2 = 49$) has the expected impact on the donor jobs (an equivalent increase for `CS-Immediate-Dispatch` and `Dedicated`), however has surprisingly little impact on the performance of the beneficiary jobs, with the noticeable impact occurring only when $\rho_B > 1$. We would have assumed the opposite: that beneficiaries would prefer less variable donor job sizes because that means less variable donor busy periods and more regular help. Increasing the beneficiary job size variability has the expected impact of more penalty to the donors. Higher beneficiary variability creates higher beneficiary response times for both algorithms (even at lower ρ_B), without affecting either algorithm's stability region.

In summary we see throughout that beneficiary jobs may benefit unboundedly from cycle stealing, regardless of donor and beneficiary variability. We also seen that the impact to donor jobs is comparatively small.

6 Analysis and results under multiple beneficiary servers

We now discuss the scenario where there are n classes of beneficiary jobs (each with its own server) and a single donor server. Beneficiary jobs of class i arrive with rate λ_i and have generally-distributed size X_{B_i}. The beneficiary jobs of class i first check if the donor server is idle and if so go there, otherwise they go to the ith beneficiary queue.

The analysis for this $n + 1$ server case is very similar to the two server case. As before, the system state can be decomposed into the donor queue and the beneficiary

queues. The donor server receives arrivals with rate λ_D when it is busy, but with rate $\lambda_D + \sum_i \lambda_{B_i}$ when it is idle. Following the same virtual waiting time analysis as for the two server case, we obtain the following results for the waiting time of a donor job:

Theorem 6.1 *Assuming n beneficiary job classes with job sizes X_{B_1},...,X_{B_n} and arrival rates are λ_{B_1},...,λ_{B_n}, respectively, the performance of donor jobs has the following representation:*

$$\begin{aligned}
\tilde{T}^{(D)}(s) &= \frac{s + \sum_{i=1}^{n} \lambda_{B_i}(1 - \tilde{X}_{B_i}(s))}{s - \lambda_D + \tilde{X}_D(s)\lambda_D} \pi_0 \cdot \widetilde{X_D}(s); \\
\mathsf{E}\left[T^{(D)}\right] &= \frac{\rho_D}{1-\rho_D} \frac{\mathsf{E}\left[X_D^2\right]}{2\mathsf{E}\left[X_D\right]} + \frac{\sum_{i=1}^{n} \rho_{B_i} \frac{\mathsf{E}\left[X_{B_i}^2\right]}{2\mathsf{E}[X_{B_i}]}}{1 + \sum_{j=1}^{n} \rho_{B_j}} \\
&\quad + \mathsf{E}\left[X_D\right]; \\
\mathsf{E}\left[T^{(D)2}\right] &= \frac{\rho_D}{1-\rho_D} \frac{\mathsf{E}\left[X_D^3\right]}{3\mathsf{E}\left[X_D\right]} + \frac{\sum_{i=1}^{n} \rho_{B_i} \frac{\mathsf{E}\left[X_{B_i}^3\right]}{3\mathsf{E}[X_{B_i}]}}{1 + \sum_{j=1}^{n} \rho_{B_j}} \\
&\quad + \frac{\rho_D}{1-\rho_D} \frac{\mathsf{E}\left[X_D^2\right]}{\mathsf{E}\left[X_D\right]} (\mathsf{E}\left[T^{(D)}\right] - \mathsf{E}\left[X_D\right]) \\
&\quad + 2\mathsf{E}\left[X_D\right] \mathsf{E}\left[T^{(D)}\right] + \mathsf{E}\left[X_D^2\right] - 2\mathsf{E}\left[X_D\right]^2;
\end{aligned}$$

where

$$\pi_0 = \frac{1 - \rho_D}{1 + \sum_{j=1}^{n} \rho_{B_i}}$$

represents the fraction of time that the donor host is idle.

Next we derive the busy period for the donor server. Similarly to the two server case we have:

$$\begin{aligned}
\widetilde{B_D}(s) &= \widetilde{X_D}(s + \lambda_D - \lambda_D \widetilde{B_D}(s)); \\
\widetilde{B_{B_i D}}(s) &= \widetilde{X_{B_i}}(s + \lambda_D - \lambda_D \widetilde{B_D}(s)).
\end{aligned}$$

for $1 \leq i \leq n$, where n is the number of beneficiary servers.

$$\begin{aligned}
\mathsf{E}[B_D] &= \frac{\mathsf{E}[X_D]}{1-\rho_D}; \qquad \mathsf{E}[B_{B_i D}] = \frac{\mathsf{E}[X_{B_i}]}{1-\rho_D}; \\
\mathsf{E}\left[B_D^2\right] &= \frac{\mathsf{E}\left[X_D^2\right]}{(1-\rho_D)^3}; \\
\mathsf{E}\left[B_{B_i D}^2\right] &= \mathsf{E}[X_{B_i}] \cdot \lambda_D \cdot \frac{\mathsf{E}\left[X_D^2\right]}{(1-\rho_D)^3} + \frac{1}{(1-\rho_D)^2} \cdot \mathsf{E}\left[X_{B_i}^2\right]; \\
\mathsf{E}\left[B_D^3\right] &= \frac{3\lambda_D (\mathsf{E}\left[X_D^2\right])^2}{(1-\rho_D)^5} + \frac{\mathsf{E}\left[X_D^3\right]}{(1-\rho_D)^4}; \\
\mathsf{E}\left[B_{B_i D}^3\right] &= \frac{\lambda_D^2 \mathsf{E}[X_{B_i}] \cdot 3 \cdot (\mathsf{E}\left[X_D^2\right])^2}{(1-\rho_D)^5} + \frac{\mathsf{E}\left[X_{B_i}^3\right]}{(1-\rho_D)^3}; \\
&\quad + \frac{3\lambda_D \mathsf{E}\left[X_D^2\right] + \mathsf{E}\left[X_{B_i}^2\right] + \lambda_D \mathsf{E}[X_{B_i}] \mathsf{E}\left[X_D^3\right]}{(1-\rho_D)^4}.
\end{aligned}$$

Again, similarly to the 2-server case, we have:

$$\mathsf{E}\left[B^\ell\right] = \frac{\sum_{i=1}^{n} \lambda_{B_i} \mathsf{E}\left[B_{B_i D}^\ell\right]}{\sum_{j=1}^{n} \lambda_{B_j} + \lambda_D} + \frac{\lambda_D}{\sum_{j=1}^{n} \lambda_{B_j} + \lambda_D} \mathsf{E}\left[B_D^\ell\right].$$

Finally we analyze the Markov chain for the ith beneficiary queue. This chain is shown in Figure 7. Observe that we only track the number of beneficiary jobs of class i, as well as tracking whether the donor server is busy or idle. The bold transition marked B will be replaced by a 2-stage Coxian which matches the first 3 moments of the donor server busy period, as computed by the above equation.

Also, the stability condition is easily extended from Theorem 4.1 as follows:

Theorem 6.2 *The stability condition for donor queue is $\rho_D < 1$, and the stability condition for the i-th beneficiary queue is*

$$\rho_D < \frac{1 + \sum_j \rho_{B_j}}{\rho_{B_i}} - \sum_j \rho_{B_j}$$

for all i.

In Figure 8 we show results for the case of 1 to 20 beneficiary classes, where the beneficiary classes and are all i.i.d. with exponentially-distributed job sizes with mean 1. There are many interesting observations to be made (some not shown on the figure for lack of space). First, observe that even with 20 beneficiaries, the response times of beneficiaries under `CS-Immediate-Dispatch` is still a big improvement upon `Dedicated`, because of the enlarged stability region. Specifically, at $\rho_B = 1$, the mean response time is *infinite* under `Dedicated`, but only 41 under `CS-Immediate-Dispatch` for each of the beneficiary classes (not viewable from figure). Furthermore, observe that the change in going from i to $i+1$ beneficiaries decreases rapidly with i: There is a big change in beneficiary performance when we increase from 1 beneficiary class to 2 beneficiary classes. However, there is almost no change at all in moving from 20 beneficiary classes to 30 beneficiary classes. Observe likewise that increasing the number of beneficiary classes only slightly effects donor performance beyond the first few beneficiaries. After that point, the donor performance is bounded by an M/G/1 with setup cost X_B (that is, an M/G/1 consisting of only donor jobs, where the first arrival to a busy period always sees a beneficiary job).

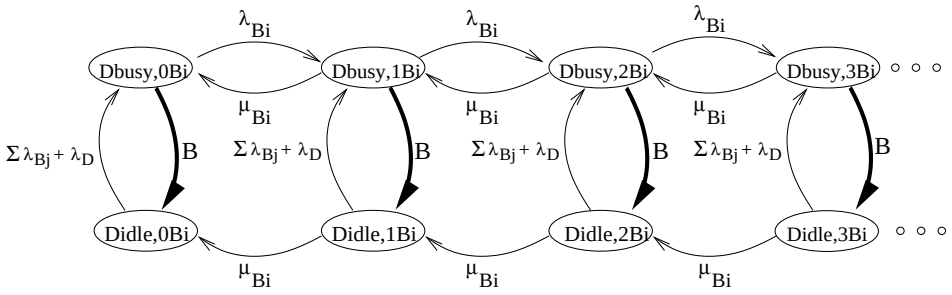


Figure 7: *MC for ith beneficiary server – shown where job sizes are exponentially-distributed. Note the similarity to Figure 3(a).*

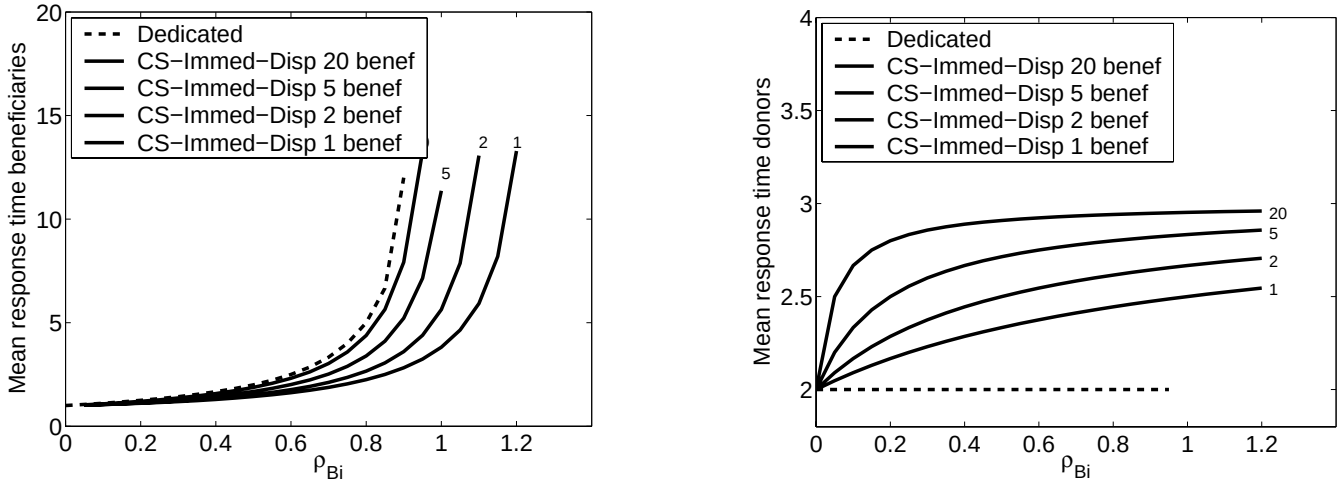


Figure 8: *Response times for the case where there are n beneficiary classes, all i.i.d.. Each curve shows a different value of n, with higher values of n leading to less gain for beneficiaries and more pain for donors. In the graph we fix $\rho_D = 0.5$. Donor and beneficiary jobs are exponentially-distributed with mean size 1.*

7 Validation of our analytical method

As we are proposing a new analytical scheme to arrive at near-exact calculations of waiting times in the system, it is of paramount importance that we demonstrate the correctness of our proposed method. In this section, we validate the accuracy of our method in two ways:

(i) Validation against known limiting cases: We compare the output of our algorithm with known exact results from the literature when these exist. Due to the complexity of our system, this is possible only in a limited number of special cases; specifically when the traffic intensity of one of the customer classes approaches either zero or the saturation point of the system.

(ii) Validation against simulation: Having evaluated our approximation methods for limiting cases, we next consider intermediate loads. Computer simulation provides an effective method for testing our analytical results over a broad range of loads, limited only by the fact that simulation accuracy decreases as the relative traffic intensities approach saturation [3, 35].

7.1 Validation against known limiting cases

We describe two limiting cases, which we use to evaluate our analysis. In both cases we assume beneficiary jobs are exponentially distributed; donor jobs are drawn from a Coxian distribution with $C^2 = 8$. Many other distributions of job sizes were evaluated; all resulted in the same limiting behavior.

Limiting Case 1: Fix ρ_B, take $\rho_D \to 1$. Under this case, under `CS-Immediate-Dispatch` it becomes increasingly difficult for the beneficiary jobs to gain access to the donor server. Thus, the mean response time for *beneficiary jobs* should approach that of an M/G/1 queue with load ρ_B. This is in fact the case, as shown in Figure 9 (row 1), where we fix $\rho_B = 0.9$ and evaluate the response time for beneficiary jobs as ρ_D is set progressively closer to 1.

Limiting Case 2: Fix ρ_D, take $\rho_B \to 0$. Under this case, under `CS-Immediate-Dispatch` it becomes increasingly unlikely that a donor job will be obstructed by a short job. Thus the mean response time for *donor jobs* should approach that of an M/G/1 queue with load ρ_D. This is in fact the case, as shown in Figure 9 (row 2), where we fix $\rho_D = 0.9$ and evaluate mean response time for donors as ρ_B is set progressively closer to 0.

7.2 Validation against simulation

We performed event-driven simulations of our `CS-Immediate-Dispatch` algorithm in C on a 700MHz Pentium III processor with 256 MB RAM. We experimented with a range of beneficiary and donor loads, mean job sizes, and variability in the job size distributions (up to $C^2 = 8$). Each experiment consisted of measuring mean response time over 10^6 arrivals with a warmup period of 50,000 arrivals. Each experiment was then replicated thirty times (using different seeds) and the average of the thirty replications was compared with the analytically-predicted value from our algorithm.

Almost all of our simulation results were within 1–2% of predicted analysis. In some cases the simulation numbers were a little higher than analysis and in some cases a little lower. Figure 10 shows just a *small subset* of our experiments, restricted to exponential job sizes, where ρ_B is held fixed at 0.9 and ρ_D is allowed to range over all stable values. Simulation replications were quite consistent at low loads and exhibited high variability at higher loads, even under an exponential job size distribution. When the job size distribution was Coxian (with $C^2 = 8$), the variation within the simulation results increased further. Due to space concerns, we do not show the Coxian plots, but we do include those results in our discussion below.

Over all our simulation experiments, we see that the discrepancy between simulation and analysis is primarily limited to the performance of the *beneficiary* jobs, under high load, in the case where beneficiaries are shorter than the donors. This can be explained as follows: The only approximation in our analysis of `CS-Immediate-Dispatch` stems from matching only the first 3 moments of the length of the busy periods. These are busy periods consisting of donor jobs, or primarily donor jobs. There is variability in the length of these busy periods, which becomes more pronounced when donor jobs are very long and loads are very high. To fully capture the effect of these busy periods, we will need to match more moments. Using more sophisticated simulation techniques to ameliorate the variability in simulation results caused by the high traffic intensity will likely help as well. One should note that over all the simulation experiments that we ran, the difference between analysis and simulation was almost always within 1–2%, with higher differences occurring only at high traffic intensity. The max difference was under 10%.

It is worth pointing out that for each graph in Figure 10, the simulation portion required close to an hour to generate, whereas the analysis portion required less than a second to compute.

8 Conclusion

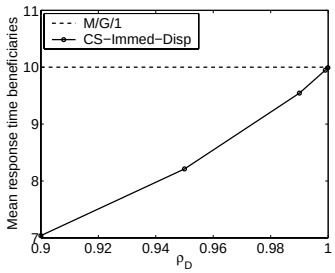
This paper presents the first analysis of task assignment with immediate dispatch and cycle stealing. It is also the first to quantify the effect of multiple beneficiaries stealing from a single donor server.

Our findings are that immediate dispatch with cycle stealing vastly improves the performance of the beneficiaries, by increasing the stability region for the system. Furthermore, the gains obtained by the beneficiaries are surprisingly insensitive to variability in donor behavior, specifically the variability in the donor service requirements. We also find that the donor jobs do not suffer much by having their idle cycles stolen.

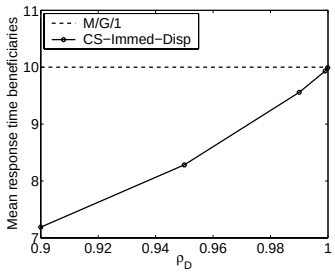
These same findings persist in the case of multiple beneficiary servers stealing from a single donor server. The reduction in the amount of benefit as more beneficiary servers are added is, interestingly, highly nonlinear, with most of the reduction coming from the first additional beneficiary. This intuition behind this observation follows from the stability analysis of the system.

In addition to the practical significance of the results, the paper also contributes an analytical method: First, the task assignment policy is defined so as to allow decomposition of the servers. Second, the paper uses "busy period transitions" to reduce the dimensionality of the beneficiary server from a 2D-infinite Markov chain (intractable) to a 1D-infinite Markov chain (tractable). The only approximation in this method is the approximation of the busy period duration by its first 3 moments (where more moments can be used to achieve any desired level of accuracy). We hope that this approach extends to the analysis of other systems problems.

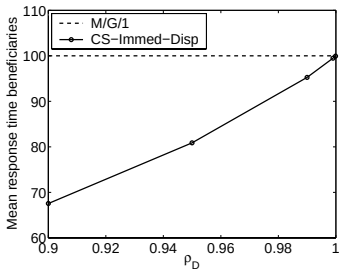
Validation against Limiting Case 1



(a) beneficiaries 1, donors 1

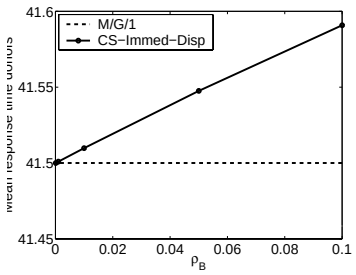


(b) beneficiaries 1, donors 10

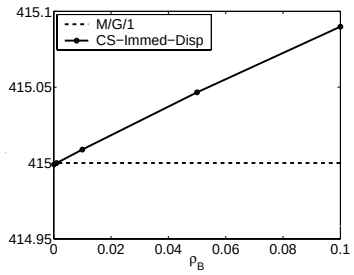


(c) beneficiaries 10, donors 1

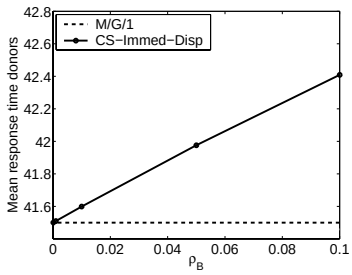
Validation against Limiting Case 2



(a) beneficiaries 1, donors 1



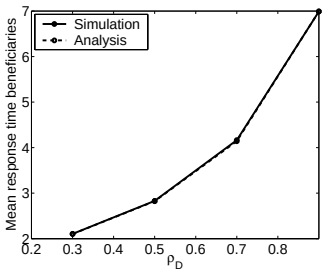
(b) beneficiaries 1, donors 10



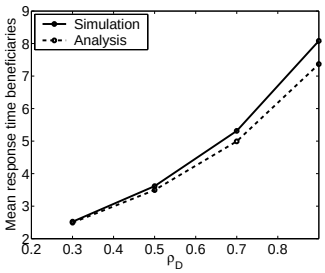
(c) beneficiaries 10, donors 1

Figure 9: *Validation of analysis against limiting cases. In row 1, $\rho_B = 0.9$. As $\rho_D \to 1$, response times of beneficiary jobs converge to an M/G/1 with load ρ_B. In row 2, $\rho_D = 0.9$. As $\rho_B \to 0$, response times for donor jobs converge to an M/G/1 with load ρ_D.*

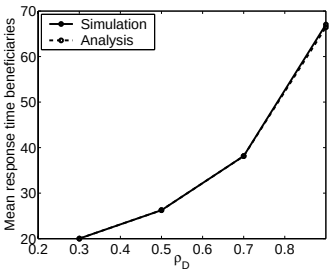
Validation against Simulation, Performance of Beneficiaries



(a) beneficiaries 1, donors 1

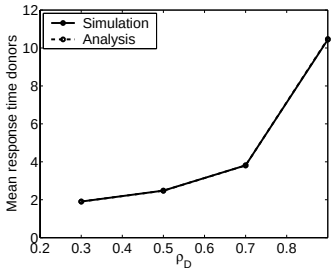


(b) beneficiaries 1, donors 10

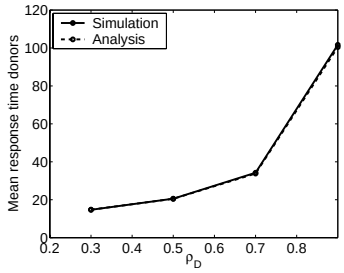


(c) beneficiaries 10, donors 1

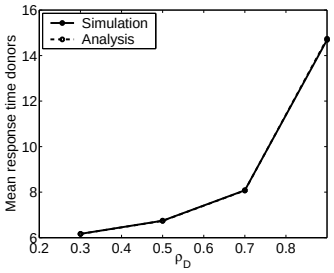
Validation against Simulation, Performance of Donors



(a) beneficiaries 1, donors 1



(b) beneficiaries 1, donors 10



(c) beneficiaries 10, donors 1

Figure 10: *Validation of analysis against simulation. Throughout we fix $\rho_B = 0.9$.*

References

[1] The PSC's Cray J90's. http://www.psc.edu/machines/cray/j90/j90.html, 1998.

[2] Supercomputing at the NAS facility. http://www.nas.nasa.gov/Technology/Supercomputing/, 1998.

[3] S. Asmussen. Queueing simulation in heavy traffic. *Mathematics of Operations Research*, 17(1), 1992.

[4] 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.

[5] F. Bonomi and A. Kumar. Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler. *IEEE Transactions on Computers*, 39(10):1232–1250, October 1990.

[6] Cisco Systems Local Director. http://www.cisco.com/warp/public/cc/pd/cxsr/400/index.shtml.

[7] D. Cox. A use of complex probabilities in the theory of stochastic processes. *Proceedings of Cambridge Philosophical Society*, 51:313 – 319, 1955.

[8] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

[9] M. E. Crovella, M. S. Taqqu, and A. Bestavros. Heavy-tailed probability distributions in the world wide web. In *A Practical Guide To Heavy Tails*, chapter 1, pages 1–23. Chapman & Hall, New York, 1998.

[10] A. Ephremides, P. Varaiya, and J. Walrand. A simple dynamic routing problem. *IEEE Transactions on Automatic Control*, AC-25(4):690–693, 1980.

[11] D. Feitelson, L. Rudolph, U. Schwiegelshohn, K. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In *Proceedings of IPPS/SPDP '97 Workshop. Lecture Notes in Computer Science, vol. 1291*, pages 1–34, April 1997.

[12] M. Harchol-Balter. Task assignment with unknown duration. *Journal of the ACM*, 49(2), 2002.

[13] M. Harchol-Balter, M. Crovella, and C. Murta. On choosing a task assignment policy for a distributed server system. *IEEE Journal of Parallel and Distributed Computing*, 59:204 – 228, 1999.

[14] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems*, 15(3), 1997.

[15] M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, and M. Squillante. Task assignment with cycle stealing under central queue. In *To appear in Proceedings of 23rd International Conference on Distributed Computing Systems (*ICDCS '03*)*., May 2003. For a copy see http://www.cs.cmu.edu/∼harchol/.

[16] S. G. Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pages 27–40. Springer-Verlag, 1996. Lecture Notes in Computer Science Vol. 1162.

[17] G. Hunt, G. Goldszmidt, R. King, and R. Mukherjee. Network dispatcher: A connection router for scalable internet services. In *Proceedings of the 7th International WWW Conference*, April 1998.

[18] V. S. Iyengar, L. H. Trevillyan, and P. Bose. Representative traces for processor models with infinite cache. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pages 62–72, February 1996.

[19] M. A. Johnson and M. F. Taaffe. An investigation of phase-distribution moment-matching algorithms for use in queueing models. *Queueing Systems*, 8:129–147, 1991.

[20] A. Lang and J. L. Arthur. Parameter approximation for phase-type distributions. In S. R. Chakravarthy and A. S. Alfa, editors, *Matrix-Analytic Methods in Stochastic Models*, pages 151–206. Marcel Dekker, 1997.

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

[22] C. Leiserson. The Pleiades alpha cluster at M.I.T.. Documentation at: //http://bonanza.lcs.mit.edu/, 1998.

[23] C. Leiserson. The Xolas supercomputing project at M.I.T.. Documentation available at: http://xolas.lcs.mit.edu, 1998.

[24] W. E. Leland and T. J. Ott. Load-balancing heuristics and process behavior. In *Proceedings of Performance and ACM Sigmetrics*, pages 54–69, 1986.

[25] J. D. C. Little. A proof of the queuing formula $L = \lambda W$. *Operations Research*, 9:383–387, 1961.

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

[27] M. F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Marcel Dekker, 1989.

[28] T. Osogami, M. Harchol-Balter, and A. Scheller-Wolf. Analysis of cycle stealing with switching cost. In *To appear in ACM Sigmetrics 2003 Conference on Measurement and Modeling of Computer Systems (*SIGMETRICS '03*)*., June 2003. For a copy see http://www.cs.cmu.edu/∼harchol/.

[29] E. W. Parsons and K. C. Sevcik. Implementing multiprocessor scheduling disciplines. In *Proceedings of IPPS/SPDP '97 Workshop. Lecture Notes in Computer Science, vol. 1459*, pages 166–182, April 1997.

[30] V. Paxson and S. Floyd. Wide-area traffic: The failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, pages 226–244, June 1995.

[31] D. L. Peterson and D. B. Adams. Fractal patterns in DASD I/O traffic. In *CMG Proceedings*, December 1996.

[32] K. W. Ross and D. D. Yao. Optimal load balancing and scheduling in a distributed computer system. *Journal of the ACM*, 38(3):676–690, July 1991.

[33] B. Schroeder and M. Harchol-Balter. Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. In *Proceedings of* HPDC 2000, pages 211–219, 2000.

[34] M. S. Squillante, D. D. Yao, and L. Zhang. Web traffic modeling and web server performance analysis. In *Proceedings of the IEEE Conference on Decision and Control*, December 1999.

[35] W. Whitt. Planning queueing simulations. *Management Science*, 35(11), 1989.

[36] W. Winston. Optimality of the shortest line discipline. *Journal of Applied Probability*, 14:181–189, 1977.

[37] R. W. Wolff. *Stochastic Modeling and the Theory of Queues*. Prentice Hall, 1989.

A Proof of Theorem 4.1

Proof: Let ρ_{hD} (respectively, ρ_{hB}) denote the load at the donor server (respectively, beneficiary server). Both these quantities must clearly be < 1.

We can deduce ρ_{hD} from the following equation:

$$\begin{aligned} \rho_{hD} &= \rho_B(1-\rho_{hD}) + \rho_D \\ \Rightarrow \rho_{hD} &= \frac{\rho_B + \rho_D}{1+\rho_B}. \end{aligned}$$

The first equality follows from the PASTA (Poisson arrival sees time average) principle which implies that the fraction of beneficiary jobs that are dispatched to the donor server is $(1-\rho_{hD})$. We therefore have the constraint that

$$\frac{\rho_B + \rho_D}{1+\rho_B} < 1 \iff \rho_D < 1.$$

Next we deduce ρ_{hB}, using the PASTA principle which implies that the fraction of beneficiary jobs that are dispatched to the beneficiary server is ρ_{hD}:

$$\rho_{hB} = \rho_B \cdot \rho_{hD} < 1,$$

or, equivalently,

$$\rho_B \cdot \frac{\rho_B + \rho_D}{1+\rho_B} < 1.$$

This is in turn equivalent to:

$$\rho_D < \frac{1}{\rho_B} + 1 - \rho_B.$$

■

B Derivation of response time of donor jobs

Let $W(t)$ be the virtual waiting time for the donor queue at time t. That is, a job arriving at the donor queue at time t would wait $W(t)$ before it starts being processed. By the PASTA (Poisson arrival sees the time average) principle, the virtual waiting time W is equal in distribution to the waiting time. Therefore, it suffices to analyze the virtual waiting time W for the derivation of the response time of donor jobs.

The following steps allow us to obtain the moments of $W = \lim_{t\to\infty} W(t)$:
(i) Set up a differential equation for $\tilde{W}(t,s)$, the Laplace transform of $W(t)$.
(ii) Let $t \to \infty$; then, $\frac{d\tilde{W}(t,s)}{dt} \to 0$, because the queue reaches the stationary state. Now, $\tilde{W}(s)$ is obtained as a function of π_0.
(iii) Evaluate $\tilde{W}(s=0)$ to obtain π_0.
(iv) Differentiate $\tilde{W}(s)$ to obtain moments of W.

(i) We first set up a differential equation for $\tilde{W}(t,s)$. For this purpose, we carefully examine the relationship between $W(t)$ and $W(t+\Delta t)$. First, suppose $W(t) \geq \Delta t$. Since the donor server is always busy between t and $t+\Delta t$, only donor jobs could arrive at the donor queue. Since the arrival process is Poisson with rate λ_D, the probability of having ≥ 1 arrival in time Δt is $\lambda_D \Delta t + o(\Delta t)$. Any such arrival will have service time X_D. Therefore,

$$W(t+\Delta t) = \begin{cases} W(t) - \Delta t & \text{with prob. } 1 - \lambda_D \Delta t + o(\Delta t), \\ W(t) + X_D - \Delta t & \text{with prob. } \lambda_D \Delta t + o(\Delta t), \\ \text{something else} & \text{with prob. } o(\Delta). \end{cases}$$

Next, suppose $0 \leq W(t) < \Delta t$. Let a random variable ϵ be the fraction of time that the donor server was idle during $(t, t+\Delta t)$ given that there were no arrivals during this interval. Let a random variable ϵ_D be the fraction of time that

the donor server was busy during this interval given that there was a donor arrival. Let a random variable ϵ_B be the fraction of time that the donor server was busy during the interval giving that there was a beneficiary arrival. Then,

$$W(t+\Delta t) = \begin{cases} 0 & \text{with prob. } 1-\lambda_D\Delta t-\lambda_B\epsilon\Delta t+o(\Delta t), \\ W(t)+X_D-\epsilon_D\Delta t & \text{with prob. } \lambda_D\Delta t+o(\Delta t), \\ X_B-\epsilon_B\Delta t & \text{with prob. } \lambda_B\epsilon\Delta t+o(\Delta t), \\ \text{something else} & \text{with prob. } o(\Delta). \end{cases}$$

Note that $0 \le \epsilon, \epsilon_D, \epsilon_B \le 1$. (See Figure 11)

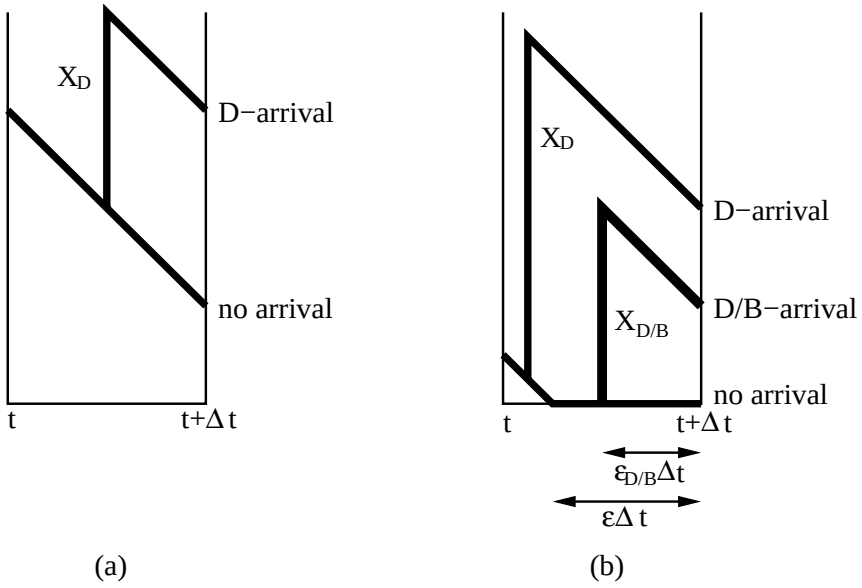


Figure 11: Virtual waiting time: relationship between $W(t)$ and $W(t+\Delta t)$, when (a) $W(t) \ge \Delta t$ and (b) $0 \le W(t) < \Delta t$.

Based on the above observation, the Laplace transform $\tilde{W}(t+\Delta t, s)$ of $W(t+\Delta t)$ is obtained as follows:

$$\begin{aligned}
&\tilde{W}(t+\Delta t, s) \\
&\quad\equiv\quad E[e^{-sW(t+\Delta t)}], \\
&\quad=\quad \int_{x=0}^{\infty} E[e^{-sW(t+\Delta t)}|W(t)=x]d\Pr(W(t)\le x), \\
&\quad=\quad \int_{x=\Delta t}^{\infty} \Big(e^{-s(x-\Delta t)}(1-\lambda_D\Delta t)+E[e^{-s(x-\Delta t+X_D)}]\lambda_D\Delta t\Big)\, d\Pr(W(t)\le x) \\
&\qquad + \int_{x=0^+}^{\Delta t} \Big((1-\lambda_D\Delta t-\lambda_B\epsilon\Delta t)+E[e^{-s(x+X_D-\epsilon_D\Delta t)}]\lambda_D\Delta t+E[e^{-s(X_B-\epsilon_B\Delta t)}]\lambda_B\epsilon\Delta t\Big)\, d\Pr(W(t)\le x) \\
&\qquad + \Big((1-\lambda_D\Delta t-\lambda_B\Delta t)+E[e^{-s(x+X_D-\epsilon_D\Delta t)}]\lambda_D\Delta t+E[e^{-s(X_B-\epsilon_B\Delta t)}]\lambda_B\Delta t\Big)\Pr(W(t)=0)+o(\Delta t), \\
&\quad=\quad \int_{x=0}^{\infty} \Big(1+(s-\lambda_D+\lambda_D\tilde{X}_D(s))\Delta t+o(\Delta t)\Big)\, e^{-sx}d\Pr(W(t)\le x) \\
&\qquad + \int_{x=0^+}^{\Delta t} \big\{1-(\lambda_D+\lambda_B\epsilon)\Delta t+E[e^{-sx}e^{-sX_D}(1+O(\Delta t))]\lambda_D\Delta t+E[e^{-sX_B}(1+O(\Delta t))]\lambda_B\epsilon\Delta t \\
&\qquad\qquad -\Big(1+(s-\lambda_D+\lambda_D\tilde{X}_D(s))\Delta t+o(\Delta t)\Big)\, e^{-sx}\big\}\, d\Pr(W(t)\le x) \\
&\qquad + \big\{(1-\lambda_D\Delta t-\lambda_B\Delta t)+E[e^{-sX_D}(1+O(\Delta t))]\lambda_D\Delta t+E[e^{-sX_B}(1+O(\Delta t))]\lambda_B\Delta t \\
&\qquad\qquad -\Big(1+(s-\lambda_D+\lambda_D\tilde{X}_D(s))\Delta t+o(\Delta t)\Big)\big\}\Pr(W(t)=0)+o(\Delta t),
\end{aligned}$$

$$= \left(1 + (s - \lambda_D + \lambda_D \tilde{X}_D(s))\Delta t\right) \tilde{W}(t,s) + \int_{x=0^+}^{\Delta t} O(\Delta t) d\Pr(W(t) \le x)$$
$$+ \left(-\lambda_B + \tilde{X}_B(s)\lambda_B - s\right) \Delta t \Pr(W(t) = 0) + o(\Delta t).$$

Thus, we obtain the next formula.

$$\frac{\tilde{W}(t+\Delta t, s) - \tilde{W}(t,s)}{\Delta t} = \left(s - \lambda_D + \lambda_D \tilde{X}_D(s)\right) \tilde{W}(t,s) + \int_{x=0^+}^{\Delta t} O(1) d\Pr(W(t) \le x)$$
$$+ \left(-\lambda_B + \tilde{X}_B(s)\lambda_B - s\right) \Pr(W(t) = 0) + \frac{o(\Delta t)}{\Delta t}.$$

Letting $\Delta t \to 0$ in the above formula, we obtain a differential equation for $\tilde{W}(t,s)$.

$$\frac{d\tilde{W}(t,s)}{dt} = \left(s - \lambda_D + \lambda_D \tilde{X}_D(s)\right) \tilde{W}(t,s) + \left(-\lambda_B + \tilde{X}_B(s)\lambda_B - s\right) \Pr(W(t) = 0).$$

(ii) Let $t \to \infty$. Then, $\frac{d\tilde{W}(t,s)}{dt} \to 0$ because the queue reaches the stationary state. Let $\tilde{W}(s) \equiv \lim_{t\to\infty} \tilde{W}(t,s)$. Then, $\tilde{W}(s)$ is obtained as a function of $\pi_0 = \Pr(W(t) = 0)$ as follows:

$$\left(s - \lambda_D + \tilde{X}_D(s)\lambda_D\right) \tilde{W}(s) = \left(s + \lambda_B - \tilde{X}_B(s)\lambda_B\right) \Pr(W(t) = 0),$$
$$\tilde{W}(s) = \frac{s + \lambda_B - \tilde{X}_B(s)\lambda_B}{s - \lambda_D + \tilde{X}_D(s)\lambda_D} \pi_0.$$

(iii) Next, we will obtain π_0 by evaluating $\tilde{W}(s)$ at $s = 0$: Note that the Laplace transform $\tilde{Z}(s)$ of a probability distribution Z always has the property $\tilde{Z}(0) = 1$.

$$1 = \tilde{W}(0),$$
$$= \left.\frac{1 - \frac{d\tilde{X}_B(s)}{ds}\lambda_B}{1 + \frac{d\tilde{X}_D(s)}{ds}\lambda_D}\right|_{s=0} \pi_0,$$
$$= \frac{1 + E[X_B]\lambda_B}{1 - E[X_D]\lambda_D} \pi_0.$$

The second equality is follows from the L'Hopital's rule. Therefore,

$$\pi_0 = \frac{1 - \lambda_D E[X_D]}{1 + \lambda_B E[X_B]}.$$

(iv) The moments of waiting time, and subsequently response time, are easily obtained by differentiating $\tilde{W}(s)$ and evaluating at $s = 0$. In particular, the n-th moment of W is

$$E[W^n] = \tilde{W}^{(n)}(0).$$