Analysis of Task Assignment with Cycle Stealing

Mor Harchol-Balter[1] Cuihong Li[2] Takayuki Osogami[3]
Alan Scheller-Wolf[4] Mark S. Squillante[5]

July 2002

CMU-CS-02-158

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

The problem of task assignment in a distributed server system is considered, where short jobs are separated from long jobs, but short jobs may be run in the long job partition if it is idle (cycle stealing). Jobs are assumed to be non-preemptible. New techniques are presented for analyzing this problem, both in the case of immediate dispatch of jobs to hosts and in the case of a central queue. The analysis is approximate, but can be made as close to exact as desired. Analysis is validated via simulation. Results of the analysis show that cycle stealing can reduce mean response time for short jobs by orders of magnitude, while long jobs are only slightly penalized.

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

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

[2]Carnegie Mellon University, GSIA. Email: cuihong@cs.cmu.edu

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

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

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

Keywords: cycle stealing, task assignment, server farm, distributed system, unfairness, starvation, load sharing, supercomputing, matrix-geometric methods, busy periods, multi-class queue, multiserver queue.

1 Introduction

Distributed server model

In recent years, distributed servers have become increasingly common because they allow for increased computing power while being cost-effective and easily scalable. In a distributed server system, jobs (tasks) arrive and must each be dispatched to exactly one of several host machines for processing. The rule for assigning jobs to host machines 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 up 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 queueing theory involve the performance analysis of task assignment policies.

In this paper we consider the *particular model* of a distributed server system in which the execution of jobs is *non-preemptive* (run-to-completion), i.e., the execution of a job can't be interrupted and subsequently resumed. We consider both the case where jobs are *dispatched immediately* upon arrival to one of the host machines for processing, and the case where jobs are held in a *central queue* until requested by a host machine.

Related applications

The model assumptions above are consistent with those of validated stochastic models used to study a wide range of high-volume Web sites [13, 25]. These papers analyze the performance of special event Web sites (1998 Winter Olympics) governed by IBM's Network Dispatcher product [12], a high-speed router which immediately routes each incoming Web requests to one of several host servers. Immediate dispatching of jobs is important in this setting for scalability and efficiency; the router must not become a bottleneck.

Our model is also motivated by servers for high-performance parallel computing. For these distributed servers, each host machine is usually a multi-processor machine (e.g., an eight-processor Cray J90) with a single memory, and each job submitted is a parallel job, intended to run on a single host. Typically, in the case of batch jobs, *one* job at a time (or a small number of jobs) occupies each host machine. Jobs may be held in a central queue until a host is ready for the job, or the jobs might be dispatched to hosts and queued at the hosts. In such settings it is typical for users to submit an upper bound on their job's CPU requirement in seconds; the job is killed if it exceeds this estimate. Examples of distributed server systems that fit our model are given in Table 1.

Our model has also been used for other applications requiring scalable systems, such as departmental computers within an organization [22] or telecommunication systems with heterogeneous servers [5]. It has also been studied simply as a purely theoretical model [24].

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.

By far the most common task assignment policy used is `Round-Robin` [13, 25, 7]. The prevalence for this policy stems from RR-DNS, [6] whereby the task assignment at a Web site is

Name	Location	No. Hosts	Host Machine
Xolas [18]	MIT Lab for Computer Science	8	8-processor Ultra HPC 5000 SMP
Pleiades [17]	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 distributed servers described by the architectural model of this paper. The schedulers used are Load-Leveler, LSF, PBS, or NQS. These schedulers typically only support run-to-completion (non-preemptive) for several reasons: First, the memory requirements of jobs tend to be huge, making it very expensive to swap out a job's memory [9]. Thus timesharing between jobs only makes sense if all the jobs being timeshared fit within the memory of the host, which is unlikely. Also, many operating systems that enable timesharing for single-processor jobs do not facilitate preemption among several processors in a coordinated fashion [21].*

done based on the IP addresses chosen by the Domain Name Server. The `Round-Robin` policy is simple, but it neither maximizes utilization of the hosts, nor minimizes response times.

In the case where the size of jobs are not known and we require immediate dispatch of jobs to hosts, then the `Shortest-Queue` task assignment policy – where incoming jobs are immediately dispatched to the host with the fewest number of jobs – has been shown to be optimal when the job size distribution is *exponential* and the arrival process is Poisson [28]. Here optimality is defined as maximizing the discounted number of jobs that complete by some fixed time t. Ephremides, Varaiya, and Walrand [8] showed that `Shortest-Queue` also minimizes the expected total time for the completion of all jobs arriving by some fixed time t, under an exponential job size distribution and arbitrary arrival process.

When a centralized queue is allowed at the dispatcher, the `M/G/k` policy been proven to minimize mean response time when job processing requirements come from an exponential distribution, or one with Increasing Failure Rate [29]. (Note: throughout this paper we will use the terms *processing requirement*, *service demand*, and *size* interchangeably.) The `M/G/k` policy holds all jobs at the dispatcher unit in a single FCFS queue, and only when a host is free does it receive the next job. This policy maximizes utilization of the host machines, thereby helping to minimize mean response time. The `M/G/k` policy is provably identical to the `Least-Work-Remaining` policy which sends each job to the host with the least total remaining work [10].

While policies like `M/G/k` 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 that the `Dedicated` policy far outperforms all these other policies with respect to minimizing mean response time. In the `Dedicated` policy, for example, some hosts may be designated as the "short hosts" and others as the "long hosts." Short jobs are always sent to the short hosts and long jobs to the long hosts. The superiority of the `Dedicated` policy is demonstrated analytically in [11], and shown empirically on several traces of supercomputing workloads in [23]. The `Dedicated` policy is also popular in practice (e.g. Cornell Theory Center) where different host 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. The intuition behind the `Dedicated` policy is that for the case of high-variability job size distributions, it is important to isolate short jobs from the long jobs, as waiting behind the long jobs is very costly.

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 [10].

Motivation for Cycle Stealing

While `Dedicated` assignment may be preferable to the `M/G/k` and `Shortest-Queue` policies for highly variable job sizes, it is clearly *not* optimal. One problem is that `Dedicated` leads to situations where the servers are not fully utilized. For example, there may be five consecutive short jobs, with no long job, resulting in an idle long host. 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-cutting benefit of the `Dedicated` policy with the high-utilization property of `M/G/k` and `Shortest-Queue`. We would like to segregate jobs by size (short jobs go to short host, long jobs go to long host) so as to provide isolation for short jobs. However during times when the long job host is free, we would like to *steal* the long host's idle cycles and use those to serve excess short jobs. This would make our system more efficient not only by decreasing the mean response time of short jobs, but also by enlarging the *stability region* of the overall system. Specifically, for highly unbalanced systems, where the short host is more heavily loaded, granting the short jobs limited access to the long partition may be the difference between an overloaded system and a well-behaved one. (We discuss this in greater detail in Section 5.) Importantly, the short jobs should only use the long host when that host is *free*, so that we don't *starve* the long jobs, or cause them undue delay. Because jobs are not preemptible, there will still be some penalty to the long jobs, because a long job may arrive to find a short job serving at the long host, causing the long job to have to wait behind that short job.

Two cycle stealing algorithms

We propose two cycle stealing algorithms:

Cycle stealing with Immediate Dispatch (`CS-Immediate-Dispatch`): In this algorithm, all jobs are immediately dispatched to a host upon arrival. There is a designated short job host and a designated long job host. An arriving long job is always dispatched to the long job host. An arriving short job first checks to see if the long job host is idle. If so, the short job is dispatched to the long job host. If, however, the long job host is not idle (either it's working on a long job or a short job), then the arriving short job is dispatched to the short job host. Jobs at a host are serviced in FCFS order. The `CS-Immediate-Dispatch` algorithm is shown in Figure 1.

The `CS-Immediate-Dispatch` algorithm is an improvement over `Dedicated`, since a fraction of the arrival stream of short jobs can be offloaded to the long host, while only slightly penalizing long jobs. However, only those short jobs arriving after the long host has entered an idle period can obtain this benefit. If a short job arrives just before the long host enters an idle period, the short job is not eligible for running during the idle partition. This is the motivation behind our next cycle stealing algorithm.

Cycle stealing with Central Queue (`CS-Central-Queue`)**:** In this algorithm, all jobs are held in a central queue. Whenever the short job host becomes idle, it picks the first short job in the central queue to run. Whenever the long job host becomes idle, it picks the first long job in the central queue to run. However, if there is no long job, the long host picks the first short job in the central queue. A minor point: Whereas in `CS-Immediate-Dispatch`, the short and long hosts are designated in advance, in `CS-Central-Queue` we allow renaming of hosts – i.e., if the long host is working on a short job, and the short host is idle, then the long host is renamed the short host and vice versa. Thus in `CS-Immediate-Dispatch`, there could be one short in front of one long job in the system with an idle (short) server, while this could not happen under `CS-Central-Queue`. The `CS-Central-Queue` algorithm is shown in Figure 2.

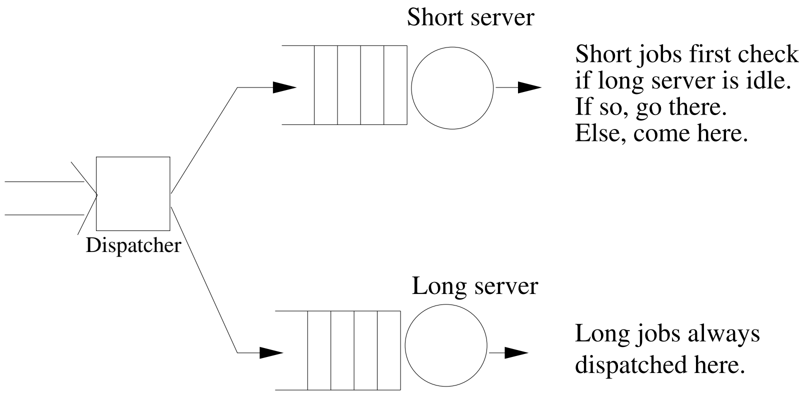


Figure 1: The `CS-Immediate-Dispatch` algorithm.

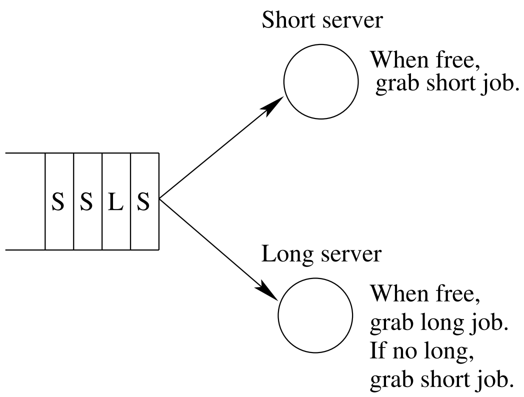


Figure 2: The `CS-Central-Queue` algorithm.

Removing the distinction between short and long

It is important to realize that the terms "short" and "long" are not meant to necessarily indicate the length of the job. There is nothing in our analysis that requires that "short" jobs be shorter than “long” jobs. These names are simply notational conveniences to indicate which jobs/host are the “donors” (providers of idle cycles) and which jobs/host are the “benefactors” (beneficiaries of the idle cycles). In fact, throughout we will consider three cases: shorts shorter than longs; shorts indistinguishable from longs; and (pathologically) shorts longer than longs. We will find that the “shorts” benefit in all three cases. The “longs” suffer little, except in the pathological case where they could get stuck waiting behind a “short” job that is not short at all. In this case the penalty to the “long” jobs is more significant, but still is dominated by the benefits to the “short” jobs.

Difficulty of analysis and new analytic approaches

The above cycle stealing idea is certainly not new. Policies like `CS-Immediate-Dispatch` and `CS-Central-Queue`, as well as others of a similar flavor, have been suggested in countless papers. However until now the analysis of such policies has eluded researchers. Observe that even for simplest instance of our problem – where job arrivals are Poisson, short jobs are drawn i.i.d. from one exponential distribution and long jobs are drawn i.i.d. from another exponential distribution – the continuous-time Markov chain, while relatively easy to describe, is mathematically intractable. This is due to the fact that the stochastic process grows infinitely in two dimensions and it contains no structure that can be exploited in order to obtain an exact solution. While 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. Thus truncation is neither sufficiently accurate nor robust for our purposes.

This paper provides the first analysis (to our knowledge) of the `CS-Immediate-Dispatch` and `CS-Central-Queue` policies. In both cases, the analysis is approximate, but can be made as close to exact as desired. Our analysis assumes a Poisson arrival process for short and long jobs. For both algorithms, the service requirements of the short and long jobs are assumed to be drawn i.i.d. from any general phase-type distribution. Our approaches for analyzing `CS-Immediate-Dispatch` and `CS-Central-Queue` are very different, but they do share one common element: busy period approximation.

For `CS-Immediate-Dispatch`, we decompose the stochastic process representing the system into two processes, one corresponding to the short host and one corresponding to the long host. We solve the stochastic process corresponding to the long host exactly. We then derive the impact of the long host on the short host and use this to determine the appropriate stochastic process for the short host, which we then solve. The only approximation lies in the impact of the long host's busy period on the short host process. This approximation can be made as precise as desired, since the long host's busy period can be estimated up to any number of moments. In this paper we match three moments and verify via simulation that this is sufficient, in Section 6.

For `CS-Central-Queue`, the above technique of decomposing stochastic processes and deriving the impact of one on the other does not work. Here we appear to need a Markov chain that is infinite in two dimensions, since we must keep track of both the number of short jobs and the number of long jobs in the system. In fact, even in the case where short and long jobs are indistinguishable, the `CS-Central-Queue` policy becomes an M/G/2/FCFS policy, which is not analytically tractable. Our solution is to instead use the transitions between states to keep track of the relevant information concerning the delay of long jobs, making our Markov chain infinite in only one dimension. This is done by modeling these transitions as certain types of busy periods, for which we can derive any number of moments. We again match the first three moments of the busy periods, and verify that this is sufficient via simulation (in Section 6.)

For both the above cycle stealing algorithms we also derive stability conditions in Section 5.

Outline

Section 2 presents preliminary notation. The analysis of `CS-Immediate-Dispatch` is given in Section 3 and the analysis of `CS-Central-Queue` is given in Section 4. Section 5 presents stability criteria for both cycle stealing algorithms. Section 6 validates the analysis of both algorithms against simulation and limiting cases. Finally Section 7 shows the results of the analysis, assessing the improvement of cycle stealing over `Dedicated` assignment, while also comparing the two cycle stealing algorithms in terms of their benefit to short jobs and penalty to long jobs.

2 Notation

Throughout we assume that short (respectively, long) jobs arrive according to a Poisson process with rate λ_S (respectively, λ_L). The size, a.k.a. service requirement, of short jobs (respectively, long jobs) is denoted by the random variable X_S (respectively, X_L). The i^{th} moment of the size of a short (respectively, long) job is therefore $E[X_S^i]$ (respectively, $E[X_L^i]$). For readability, we will usually start out by showing the case where job sizes are exponentially distributed, and will use μ_S (respectively μ_L) to denote the service rate of short (respectively, long) jobs, where $\mu_S = 1/E[X_S]$ and $\mu_L = 1/E[X_L]$. We define ρ_S (respectively, ρ_L) to be the load created by short jobs (respectively, long jobs), where $\rho_S = \lambda_S \cdot E[X_S]$ and $\rho_L = \lambda_L \cdot E[X_L]$. We assume that the first three moments of the busy periods are finite, and the queues are stable. We discuss stability in greater depth in Section 5.

The Laplace Transform $L_f(s)$ of a non-negative Lebesgue integrable function $f(t)$ (on the positive real axis) is defined as

$$L_f(s) =_{dist} \int_0^\infty e^{-st} f(t) dt, \; s \geq 0.$$

The Laplace Transform of a continuous R.V. X, often denoted by $\tilde{X}(s)$ refers to the Laplace transform $L_f(s)$, of its pdf $f(x)$, in which case $L_f(s) = E[e^{-sX}]$.

3 Cycle Stealing under Immediate Dispatch

The `CS-Immediate-Dispatch` algorithm is shown in Figure 1. Under `CS-Immediate-Dispatch`, all long jobs are immediately dispatched to the long host upon arrival. When a short job arrives, it is dispatched to the long host if that host is idle. Otherwise, it is dispatched to the short host.

Our analytic approach involves a three step process. The first step is to derive the mean response times for the long jobs only. We do this in two different ways, first using Matrix Geometric methods and then using virtual waiting time analysis. We confirm that both methods yield the same result.

Figure 3(a) depicts a Markov chain describing the long host, under the assumption that job sizes are exponentially-distributed. Each state represents the number of long jobs and the number of short jobs. Figure 3(b) is the same chain generalized to the case of phase-type job sizes (a two-phase Coxian is shown). Observe that the number of short jobs at the long host is either 0 or 1. Long

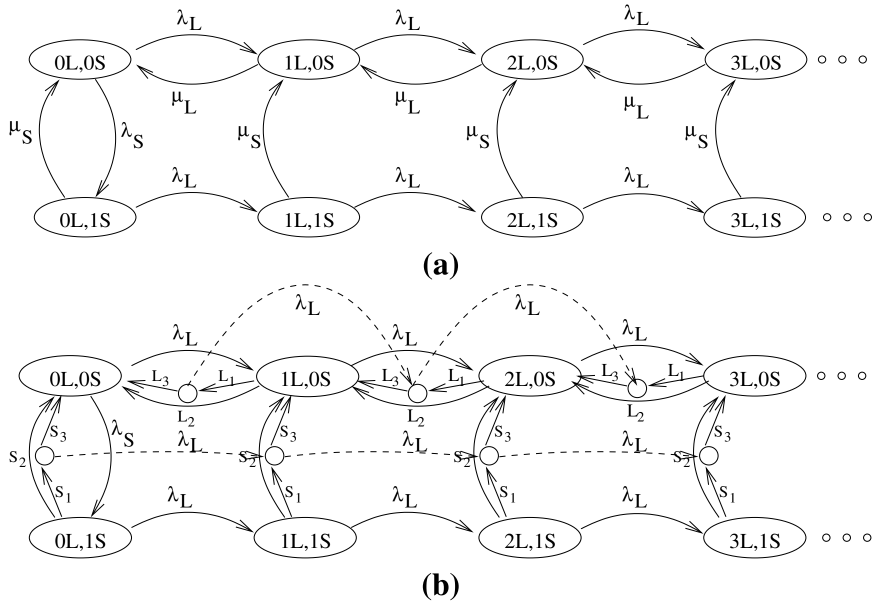


Figure 3: Markov chain for long host. (a) Shown where job sizes are exponentially-distributed. (b) Shown where job sizes have a phase-type distribution (two-phase Coxian).

jobs arrive with rate λ_L at every state. Short jobs arrive only when the long host is idle. We solve the chain in Figure 3(b) to obtain the steady-state mean number of long jobs exactly using Matrix Geometric methods. We then apply Little's Law [19] to obtain the mean response time of long jobs.

The Matrix Geometric method is due to Neuts [20] and described also in [16]. It 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 plot in this paper which used Matrix Geometric analysis (to solve multiple instances with different parameter values) was produced within a couple of seconds using the Matlab 6 environment.

An alternative way to deduce the mean response time for the long jobs is to use virtual waiting time analysis. We include this analysis in the Appendix. Below we simply state its result. $T^{(L)}$ represents the response time of long jobs.

$$\begin{aligned}
\widetilde{T}^{(L)}(s) &= \frac{s + \lambda_S - \widetilde{X}_S(s)\lambda_S}{s - \lambda_L + \widetilde{X}_L(s)\lambda_L} \cdot \pi_0 \cdot \widetilde{X_L}(s), \\
E[T^{(L)}] &= \frac{\rho_L}{1-\rho_L}\frac{E[X_L^2]}{2E[X_L]} + \frac{\rho_S}{1+\rho_S}\frac{E[X_S^2]}{2E[X_S]} + E[X_L],
\end{aligned}$$

where

$$\pi_0 = \frac{1-\rho_L}{1+\rho_S},$$

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

The second step is to derive all moments for the busy and idle periods of the long host, for use in our analysis of the short host. The long host idle time is exponentially-distributed with rate $\lambda_L + \lambda_S$. To derive the length of the busy period for the long host, we need to distinguish between two types of busy periods: (1) A busy period made up entirely of long jobs, the duration of which we represent by the r.v. B_L; and (2) A busy period started by one short job followed by zero or more long jobs, the duration of which we represent by the r.v. B_{SL}. We then have the following:

$$\begin{aligned}
\widetilde{B_L}(s) &= \widetilde{X_L}(s + \lambda_L - \lambda_L \widetilde{B_L}(s)); \\
\widetilde{B_{SL}}(s) &= \widetilde{X_S}(s + \lambda_L - \lambda_L \widetilde{B_L}(s)).
\end{aligned}$$

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

$$\begin{aligned}
\mathbf{E}\{B_L\} &= \frac{\mathbf{E}\{X_L\}}{1-\rho_L}; \qquad \mathbf{E}\{B_{SL}\} = \frac{\mathbf{E}\{X_S\}}{1-\rho_L}; \\
\mathbf{E}\left\{B_L^2\right\} &= \frac{\mathbf{E}\left\{X_L^2\right\}}{(1-\rho_L)^3}; \\
\mathbf{E}\left\{B_{SL}^2\right\} &= \mathbf{E}\{X_S\} \cdot \lambda_L \cdot \frac{\mathbf{E}\left\{X_L^2\right\}}{(1-\rho_L)^3} + \frac{1}{(1-\rho_L)^2} \cdot \mathbf{E}\left\{X_S^2\right\}; \\
\mathbf{E}\left\{B_L^3\right\} &= \frac{3\lambda_L (\mathbf{E}\left\{X_L^2\right\})^2}{(1-\rho_L)^5} + \frac{\mathbf{E}\left\{X_L^3\right\}}{(1-\rho_L)^4}; \\
\mathbf{E}\left\{B_{SL}^3\right\} &= \frac{\lambda_L^2 \mathbf{E}\{X_S\} \cdot 3 \cdot (\mathbf{E}\left\{X_L^2\right\})^2}{(1-\rho_L)^5} + \frac{\mathbf{E}\left\{X_S^3\right\}}{(1-\rho_L)^3}; \\
&\quad + \frac{3\lambda_L \mathbf{E}\left\{X_L^2\right\} + \mathbf{E}\left\{X_S^2\right\} + \lambda_L \mathbf{E}\{X_S\} \mathbf{E}\left\{X_L^3\right\}}{(1-\rho_L)^4}.
\end{aligned}$$

To obtain the moments of a general busy period for the long host, observe that a busy period is of type B_{SL} with probability $\lambda_S/(\lambda_S + \lambda_L)$ and of type B_L with probability $\lambda_L/(\lambda_S + \lambda_L)$. Hence, denoting by B the duration of a general busy period, we have

$$\mathbf{E}\left\{B^\ell\right\} = \frac{\lambda_S}{\lambda_S + \lambda_L} \mathbf{E}\left\{B_{SL}^\ell\right\} + \frac{\lambda_L}{\lambda_S + \lambda_L} \mathbf{E}\left\{B_L^\ell\right\}.$$

We now construct a phase-type distribution to match as many moments of the long host's busy period as are of interest. Many methods exist for matching moments to phase-type distributions: [14, 4, 15]. We find that simply fitting a two-phase Coxian distribution to the first three moments of the long host's busy period works sufficiently well for our purposes.

Our last step is to analyze the short host. The arrival rate at the short host is λ_S only during times when the long host is busy. When the long host is idle, the arrival rate at the short host is zero. To represent the short host, we therefore need a way of representing the duration of a busy period at the long host. Figure 4(a) is a simplified view of the Markov chain model of the short host. It assumes that the job sizes are exponentially-distributed and shows the busy period duration for the long host is shown as a bold transition marked B. Figure 4(b) is the Markov chain that we actually solve for the short host. Here the job sizes are drawn from a two-phase Coxian, used to match the first three moments of the respective job size distribution. Also the length of the busy period for the long host is matched for the first three moments by a two-phase Coxian distribution.

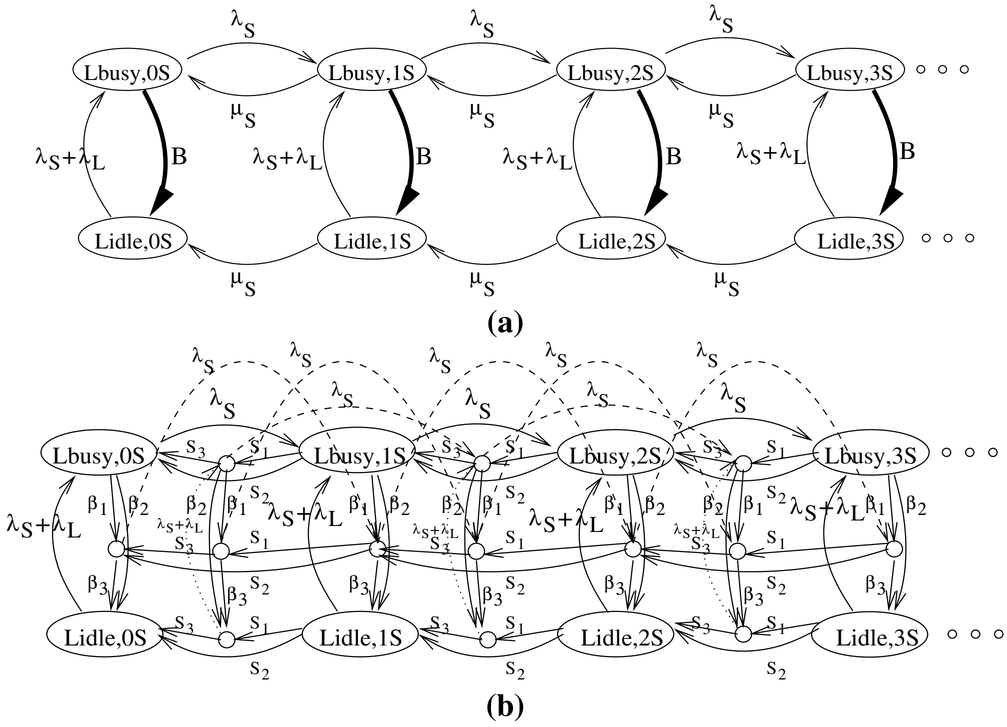


Figure 4: MC for short host. (a) Where job sizes are exponentially-distributed. (b) Where job sizes are Coxian.

We solve the Markov chain in Figure 4(b) for the number of short jobs at the short host using Matrix Geometric methods. Then via Little's Law, we obtain the mean response time of small jobs which serve at the short host. Aggregating short jobs at both hosts, we then have by PASTA:

$$
\begin{aligned}
&E[\text{ Time for short jobs }] = \\
&\qquad \Pr\{\text{Long host idle}\} \cdot E[X_S] + \Pr\{\text{Long host busy}\} \cdot E[\text{ Time at short host}]
\end{aligned}
$$

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

4 Cycle Stealing under Central Queue

The `CS-Central-Queue` algorithm is shown in Figure 2. Under `CS-Central-Queue`, all jobs are kept in a central queue in the order they arrive. When the short host is free, it grabs the next short job and runs it. When the long host is free, it grabs the next long job and runs it. If there is no long job, it grabs the next short job.

Observe that the technique we used for solving `CS-Immediate-Dispatch` (where we split the problem into two chains where one was a function of the other) will not work here. The reason

Notation	Definition
B_L	Busy period consisting of only long jobs, and started by a single long job (of size X_L).
E	exponential random variable with rate $2\mu_S$
B_E	Busy period consisting of only long jobs, and started by a single job of size E.
B_{E+L}	Busy period consisting of only long jobs, and started by a job of size $X_L + E$.
$N = A_E$	Number of long jobs which arrive during E
B_{N+1}	Busy period consisting of only long jobs, and started by a job whose size is the sum of $N + 1$ long jobs.

Table 2: *Notation necessary for analyzing* `CS-Central-Queue`

is that the decision of which job the long host takes depends on knowing the exact number of long and short jobs in the system: There is no way to decompose the system into two chains. We will instead look for a *single* Markov chain which represents this system and yet is infinite in only one dimension.

There will be a lot of notation in this section. We will introduce it as we go along. For the reader's convenience, the notation is summarized in Table 2 below.

4.1 Formulating the Markov chain for `CS-Central-Queue`

In Figure 5(a) we show a chain representing `CS-Central-Queue`. The first entry of each state denotes the number of small jobs, which ranges from zero to infinity. The second entry of each state denotes the number of long jobs. This is either $0L$, $1L$, or a special state denoted by $(N + 1)L$. The third entry of each state, when present, denotes the type of job in service at the long host. The service time for the long job is assumed to be generally-distributed. For simplicity in specifying the Markov chain, the service time for the small job is assumed to be exponential, with rate μ_S, although this is straightforward to generalize.

The logic behind the chain is as follows: Let's start in region 1, and consider a long arrival. This causes a transition to region 3, and that long arrival starts a long busy period, whose length is denoted by B_L. We will return to regions 1 or 2 only after time B_L. During this busy period, many smalls can arrive. These will only be served at rate μ_S since the long host is occupied by a long job.

Next consider a long arrival into region 2. This causes a transition into region 5. Before the long arrival can run, it must first wait for one of the small jobs to finish, since the small jobs are occupying both hosts. When one of the small jobs completes, the long job will move to occupy that host (that host will be renamed the long host) and we move to region 4. Thus, we sit in region 5 for time $E \sim Exp(2\mu_S)$, waiting for one of the small jobs to complete. During time E, N new long jobs arrive. At the moment one of the short jobs completes there are $N + 1$ long jobs in the system, and a long starts service. We now enter a busy period consisting of long jobs, started by a job of size $\sum_{i=1}^{N+1} X_L^{(i)}$. The length of this busy period is denoted by B_{N+1}. At the end of this busy period, we return to region 1 or 2. During our time in region 4, small jobs can of course arrive and depart. Such small jobs are served with rate μ_S since there is a long job occupying one host.

Observe that the only time that the small jobs are served at rate $2\mu_S$ is when there are zero long jobs in the system, or when a long job finds that there are already 2 small jobs serving.

The chain in Figure 5(a) uses two types of busy period transitions, denoted by B_L and B_{N+1}. We can derive all the moments of these busy periods and represent the busy period transitions by a Coxian distribution. Figure 5(b) is identical to Figure 5(a), except that the busy period transitions have been replaced by two-phase Coxian distributions, which allows us to model the first three moments of each busy period. More moments could be modeled using a higher-degree Coxian, however we will see that three moments provide sufficient accuracy.

There is still one problem: The chain in Figure 5 is not a strict Markov chain. The problem is that E, the time required to transition from region 5 to region 4, is not independent of B_{N+1}, the time required to transition from region 4 to region 1 or 2. This is because the number of long arrivals during E, N, is correlated with E and with B_{N+1}. For Figure 5 to be a Markov chain, we would need to have E and B_{N+1} be independent.

To address this problem, we define a new random variable, B_E. B_E is the length of a busy period consisting of only long jobs, started by a job of size E. Next define B_{E+L} to be the length of a busy period consisting of only long jobs, started by a job of size $E + X_L$. Observe that

$$B_{E+L} =_{dist} B_E + B_L$$

Observe also that B_{E+L} represents the total time required to transition from region 5 to region 1 or 2, and that B_{E+L} is exact – the dependencies between E and B_{N+1} are inherent in it.

We could imagine creating a single transition from region 5 to region 1 or 2 of duration B_{E+L}. This would not suit our needs, however: While it is not difficult to compute the moments of B_{E+L} and create a Coxian random variable representing B_{E+L}, using a single transition from region 5 to region 1 or 2 would not capture the fact that the rate at which small jobs are served is initially $2\mu_S$ but later μ_S.

Therefore we choose instead to express B_{E+L} as a sum of two independent random variables E and U where

$$B_{E+L} = E + U$$

Given the moments of B_{E+L} and the moments of E, we will be able to then derive the moments of U as follows:

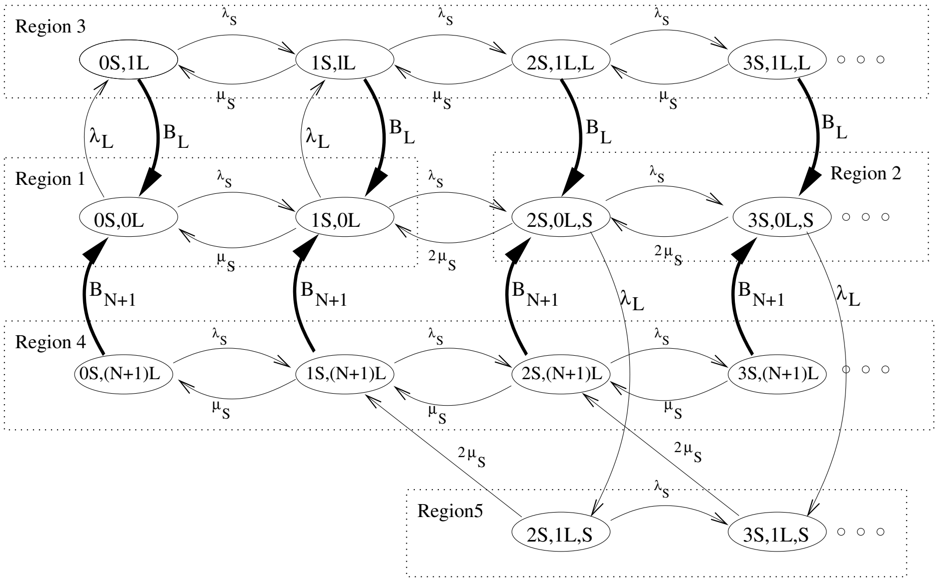
We first write out the following equations, which assume E and U are independent:

$$\begin{aligned}
B_{E+L} &= E + U, \\
E[B_{E+L}] &= E[E] + E[U], \\
E[B_{E+L}^2] &= E[E^2] + 2E[E]E[U] + E[U^2], \\
E[B_{E+L}^3] &= E[E^3] + 3E[E^2]E[U] + 3E[E]E[U^2] + E[U^3].
\end{aligned}$$

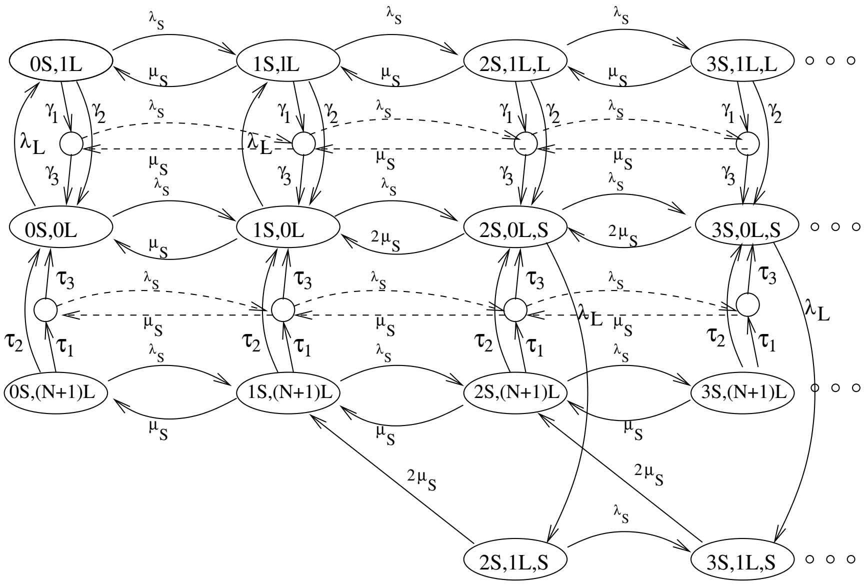
Next we solve these equations for the moments of U as follows:

$$\begin{aligned}
E[U] &= E[B_{E+L}] - E[E], \\
E[U^2] &= E[B_{E+L}^2] - E[E^2] - 2E[E]E[U], \\
E[U^3] &= E[B_{E+L}^3] - 3E[E^2]E[U] - 3E[E]E[U^2] - E[E^3].
\end{aligned}$$

We now find a two-phase Coxian which matches these first three moments of U. The resulting Markov chain, shown in Figure 6, is almost identical to Figure 5, except that the Coxian B_{N+1} random variable has been replaced by the random variable U where U is defined according to the above equations and is independent of $E \sim Exp(2\mu_S)$. Thus we have a well-defined Markov Chain. Note: Computationally, performance-wise the chain in Figure 5 is *virtually identical* to the chain in Figure 6.



(a)



(b)

Figure 5: *(a) Chain corresponding to* `CS-Central-Queue`. *The notation iS, jL represents i short jobs and j long jobs. The third field in the state denotes whether a short job or a long job is in service at the long host. Light arrows represent exponential rates. Bold arrows represent busy periods. B_L is a busy period consisting of only long jobs, and started by a single long job. B_{N+1} is a busy period consisting of only long jobs and started by $N+1$ long jobs, where N is the number of long arrivals during $Exp(2\mu_S)$. (b) Expanded version of chain in (a) where busy periods have been replaced by Coxian distributions.*

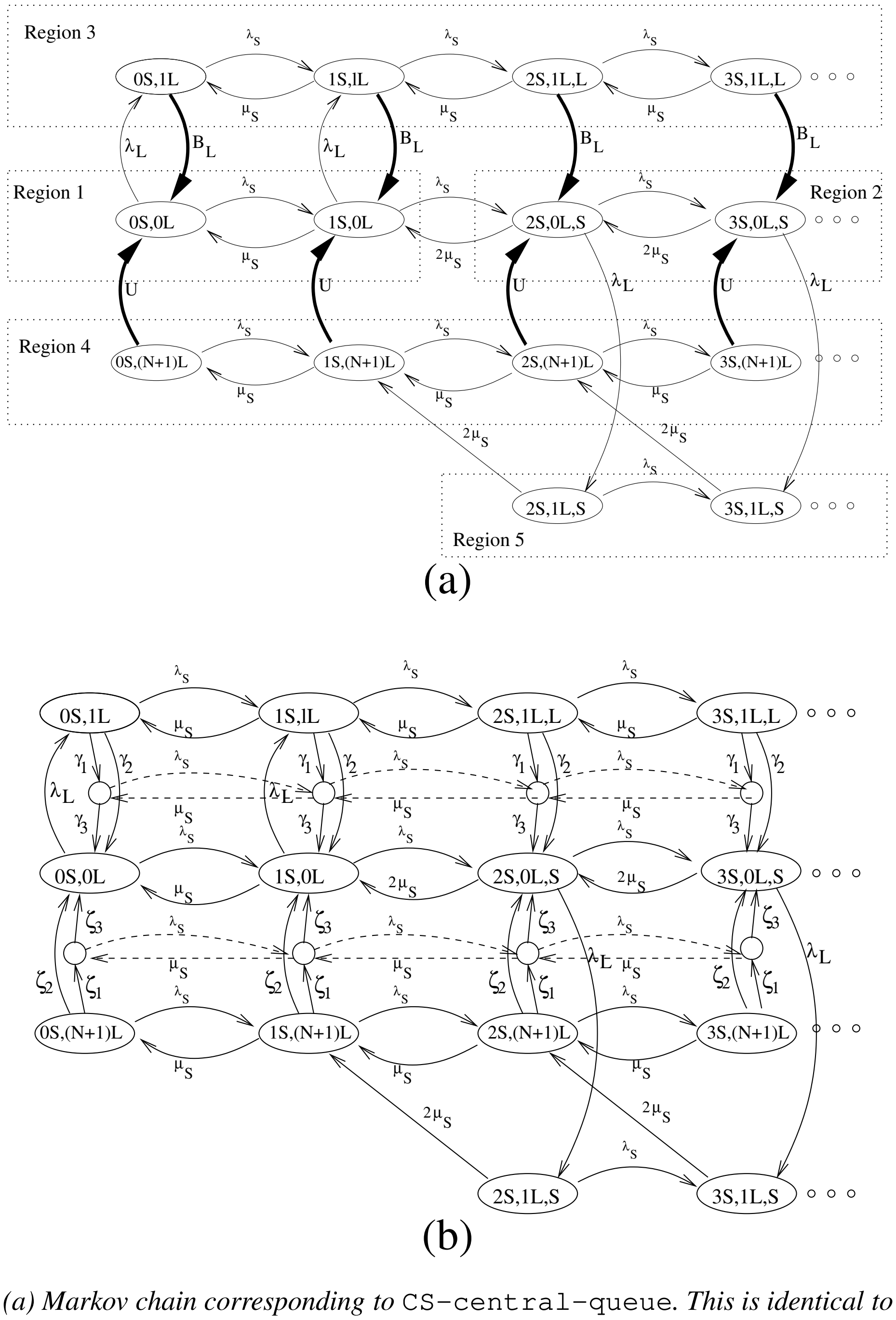


Figure 6: *(a) Markov chain corresponding to* `CS-central-queue`*. This is identical to the chain in Figure 5(a), except that the transition B_{N+1} has been replaced by U where U is independent of $E \sim Exp(2\mu_S)$. (b) Same Markov chain as in (a), but bold transitions have been replaced by two-phase Coxians matching the first three moments.*

4.2 Deriving the necessary busy periods for the chain

In order to specify the Markov chain, we need to compute the first three moments of B_L and the first three moments of B_{E+L}. These will in turn give us the first three moments of U. In this section we complete these derivations:

B_L simply represents a busy period made up of only long jobs. Its moments were computed in Section 3 and are repeated below for reference:

$$\begin{aligned} E[B_L] &= \frac{E[X_L]}{1-\rho_L}, \\ E[B_L^2] &= \frac{E[X_L^2]}{(1-\rho_L)^3}, \\ E[B_L^3] &= \frac{E[X_L^3]}{(1-\rho_L)^4} + \frac{3\lambda_L(E[X_L^2])^2}{(1-\rho_L)^5}. \end{aligned}$$

Next we derive the moments of B_E where B_E represents the length of a busy period started by a job of size $E \sim Exp(2\mu_S)$, and consisting of only long jobs.

$$\widetilde{B_E}(s) = \widetilde{E}(s + \lambda_L - \lambda_L \widetilde{B_L}(s)).$$

Therefore

$$\begin{aligned} E[B_E] &= E[E] + \lambda_L E[B_L] E[E], \\ E[B_E^2] &= E[E^2] + \lambda_L E[E] E[B_L^2] + (\lambda_L E[B_L])^2 E[E^2] + 2\lambda_L E[B_L] E[E^2], \\ E[B_E^3] &= E[E^3] + \lambda_L E[E] E[B_L^3] + 3\lambda_L^2 E[E^2] E[B_L^2] E[B_L] + (\lambda_L E[B_L])^3 E[E^3], \\ &\quad +3\lambda_L E[E^2] E[B_L^2] + 3(\lambda_L E[B_L])^2 E[E^3] + 3\lambda_L E[B_L] E[E^3]. \end{aligned}$$

Finally, we compute the moments of B_{E+L}:

$$\begin{aligned} B_{E+L} &= B_E + B_L, \\ E[B_{E+L}] &= E[B_E] + E[B_L], \\ E[(B_{E+L})^2] &= E[B_E^2] + 2E[B_E]E[B_L] + E[B_L^2], \\ E[(B_{E+L})^3] &= E[B_E^3] + 3E[B_E^2]E[B_L] + 3E[B_E]E[B_L^2] + E[B_L^3]. \end{aligned}$$

4.3 Deriving the performance of short and long jobs

Response time for short jobs

The expected response time for short jobs is easy to compute: Simply derive the limiting probabilities for the chain shown in Figure 6. The chain is analyzable via Matrix-Analytic methods, since it is infinite in only one dimension and repeats in structure starting at the third column. Its generator matrix has the form:

$$Q = \begin{pmatrix} B00 & B01 & - & - \\ B10 & B11 & A12 & - \\ - & A21 & A1 & A0 \\ - & - & A2 & A1 \end{pmatrix}.$$

From the limiting probabilities we deduce the expected number of short jobs in the system. Finally we divide by λ_s to get the mean response time for short jobs.

Response time for long jobs

For long jobs, computing the expected response time is a little more involved. From the long jobs' perspective, they see an M/G/1 queue where, at times, the first job in a busy period must wait until a short job finishes (as short jobs occupy both servers). More succinctly, assuming that short job service requirements are exponentially-distributed with rate μ_S, the response time for long jobs is the response time under an M/G/1 queue with setup time S where

$$S = \begin{cases} 0 & \text{with probability } a_1 = \frac{\Pr\{\text{Region 1}\}}{\Pr\{\text{Region 1 or 2}\}}, \\ \\ E = Exp(2\mu_S) & \text{with probability } a_2 = \frac{\Pr\{\text{Region 2}\}}{\Pr\{\text{Region 1 or 2}\}}. \end{cases}$$

Observe that S is defined entirely by what the first job to start a busy period sees. The expected waiting time for an M/G/1 queue with only long jobs and a setup time of S is known [26]:

$$E[W]^{M/G/1/SetupS} = \frac{2E[S] + \lambda_L E[S^2]}{2(1 + \lambda_L E[S])} + \frac{\lambda_L E[X_L^2]}{2(1 - \rho_L)}.$$

We thus have:

$$E[\text{Time for long}] = E[X_L] + E[W]^{M/G/1/SetupS}.$$

5 Stability under CS-Immediate-Dispatch and CS-Central-Queue

For `Dedicated` assignment it is required that $\rho_L < 1$ and $\rho_S < 1$, where ρ_L (respectively, ρ_S) denotes the load made up of long jobs (respectively, short jobs). For `CS-Immediate-Dispatch` we will see that the region of stability is much wider, and for `CS-Central-Queue` wider still.

Let ρ_{hL} (respectively, ρ_{hS}) denote the load at the long host (respectively, short host). Both these quantities must clearly be < 1.

For `CS-Central-Queue` stability conditions are trivial:

$$\begin{aligned} \rho_L &< 1, \\ \rho_S &< 2 - \rho_L. \end{aligned}$$

Therefore the rest of this section will be devoted to analyzing the stability conditions for `CS-Immediate-Dispatch`.

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

$$\begin{aligned} \rho_{hL} &= \rho_S(1-\rho_{hL}) + \rho_L, \\ \Rightarrow \rho_{hL} &= \frac{\rho_S+\rho_L}{1+\rho_S}. \end{aligned}$$

We therefore have the constraint that

$$\frac{\rho_S+\rho_L}{1+\rho_S} < 1 \iff \rho_L < 1,$$

just as in `Dedicated` and `CS-Immediate-Dispatch`. Next we deduce ρ_{hS}:

$$\rho_{hS} = \rho_S \cdot \rho_{hL} < 1,$$

or, equivalently,

$$\rho_S \cdot \frac{\rho_S+\rho_L}{1+\rho_S} < 1.$$

This is in turn equivalent to:

$$\rho_L < \frac{1}{\rho_S} + 1 - \rho_S.$$

So that ultimately it it is required that

$$\rho_L < \min(1, \frac{1}{\rho_S} + 1 - \rho_S).$$

The above constraint on ρ_L is depicted graphically in Figure 7.

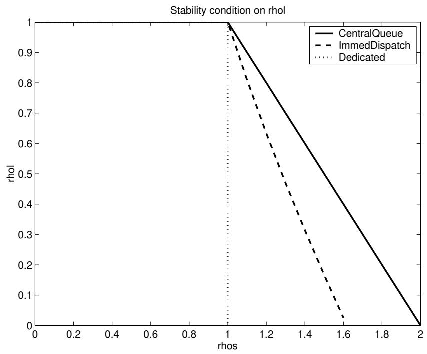


Figure 7: Stability constraint on ρ_L for `Dedicated`, `CS-Immediate-Dispatch`, and `CS-Central-Queue`

We can also turn the above constraint into a constraint on ρ_S as follows:

$$\rho_S < \frac{1-\rho_L+\sqrt{(1-\rho_L)^2+4}}{2}.$$

The restriction on ρ_S for each of the three task assignment policies is shown in Figure 8. Observe the advantage of cycle stealing in extending the stability region. When ρ_L is near zero, ρ_S can be as high as about 1.6 under `CS-Immediate-Dispatch` and as high as 2 under `CS-Central-Queue`.

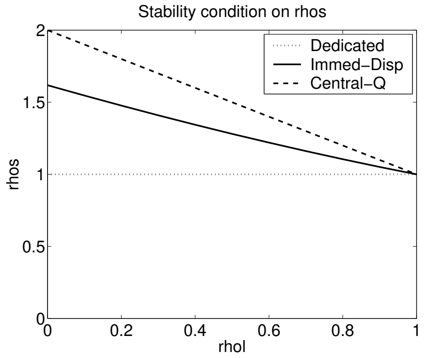


Figure 8: Stability constraint on ρ_S for `Dedicated`, `CS-Immediate-Dispatch`, and `CS-Central-Queue`.

6 Validation of Analysis

As we are proposing a new analytical scheme to arrive at near-exact calculations of waiting times, it is of paramount importance that we demonstrate the correctness of our proposed method. In both the analysis of `CS-Immediate-Dispatch` and `CS-Central-Queue`, we approximate the length of a busy period, consisting of long jobs or primarily long jobs, by its first three moments. In this section, we validate the accuracy of our methods in two ways:

1. **Validation against known limiting cases:** We compare the output of our algorithm with 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 (this saturation point may be strictly greater than one for the small jobs who have access to a second server). Depending on the model, and whether the traffic intensity approaches zero or saturation, the system approaches either an M/G/1 queue, an M/G/1/queue with initial setup time, or an M/G/2 queue. The performance of the first two of these models for general service times are known, while the third is only available in the literature only for exponential service times. Given these known results, these comparisons may be carried out for any stable traffic intensity.

2. **Validation against simulation:** Having evaluated our approximation methods for limiting cases approaching saturation and approaching zero, we next use computer simulation to test our analytical results over a broad range of loads. Simulations are limited only by the fact that simulation accuracy decreases as the relative traffic intensities approach saturation [3, 27].

For both types of validation we consider three definitions of "short" and "long."

- Shorts have mean size 1; Longs have mean size 1
- Shorts have mean size 1; Longs have mean size 10
- Shorts have mean size 10; Longs have mean size 1

Although our cycle-stealing algorithms were designed with the second case above in mind (short jobs shorter than long jobs), there is no reason why the analysis shouldn't work as well for the first and third cases above, and thus for completeness we validate under all cases.

6.1 Validation against known limiting cases

We have four limiting cases, described below, which we use to evaluate our analysis. In all cases illustrated below we assume that short jobs are exponentially-distributed and that long jobs are drawn from a Coxian distribution with $C^2 = 8$. Other distributions of job sizes and loads were also evaluated, all showing similar limiting behavior.

Limiting Case 1: Fix ρ_S, take $\rho_L \to 1$. Under this case, for both `CS-Immediate-Dispatch` and `CS-Central-Queue` it becomes increasingly difficult for the short jobs to gain access to the long server. Thus, the mean response time for *short jobs* under both cycle-stealing algorithms should approach that of an M/G/1 queue with load ρ_S. This is in fact the case, as shown in Figure 9(row 1), where we fix $\rho_S = 0.9$ and evaluate both cycle stealing algorithms with respect to mean response time for short jobs as ρ_L is set progressively closer to 1. Observe that `CS-Immediate-Dispatch` is a little faster to converge than `CS-Central-Queue`. This is because more short jobs sneak into the long server under `CS-Central-Queue`.

Limiting Case 2: Fix ρ_L, take $\rho_S \to 0$. Under this case, for both `CS-Immediate-Dispatch` and `CS-Central-Queue` it becomes increasingly unlikely that a long job will be obstructed by a short job. Thus the mean response time for *long jobs* under both cycle-stealing algorithms should approach that of an M/G/1 queue with load ρ_L. This is in fact the case, as shown in Figure 9(row 2), where we fix $\rho_L = 0.9$ and evaluate both cycle stealing algorithms with respect to mean response time for long jobs as ρ_S is set progressively closer to 0. Observe that this time `CS-Central-Queue` is a little faster to converge than `CS-Immediate-Dispatch`. To understand why, we need to consider how many short jobs interfere with long jobs. Consider times when the long host is idle. Observe that as $\rho_S \to 0$, the short host is usually idle. A new arrival under `CS-Central-Queue` will go to either queue and that will be named the short queue. However a new arrival under `CS-Immediate-Dispatch` will always try to go to the long host first (and there is no renaming). Thus fewer short arrivals will interfere with a long job under `CS-Central-Queue`, than under `CS-Immediate-Dispatch`.

Limiting Case 3: Fix ρ_L and let $\rho_S \to$ stability-condition under CS-Central-Queue. The mean response time for *long jobs* should approach an M/G/1 queue where the first job in a busy period experiences a setup cost S, where $S = min(Excess(X_S), Excess(X_S))$. The reason is that an arriving long job which sees no other long jobs, will likely see both servers occupied by short jobs. If the short job at the long host finishes first, the long host will go there. If the short job at the short host finishes first, the long job will go there and the short host will be renamed to be the long host. In the case where X_S is exponential, as we have assumed, $Excess(X_S) = X_S$ and thus we should have simply: $S = min(X_S, X_S)$. This is in fact the case, as shown in Figure 9(row 3), where we hold ρ_L fixed arbitrarily at $\rho_L = .6$ and set ρ_S progressively closer to $2 - \rho_L = 1.4$, the stability condition for `CS-Central-Queue`.

Limiting Case 4: Let $\rho_L \to 0$ under CS-Central-Queue. Under CS-Central-Queue, as $\rho_L \to 0$, the mean response time for *short jobs* should approach an M/M/2 queue. This is in fact the case, as shown in Figure 9(row 4), where we hold ρ_S fixed at 1.89, and set ρ_L progressively closer to zero.

Having validated cases with traffic intensity approaching zero or saturation, we next consider intermediate values of traffic intensity. These are evaluated via simulation.

6.2 Validation against simulation

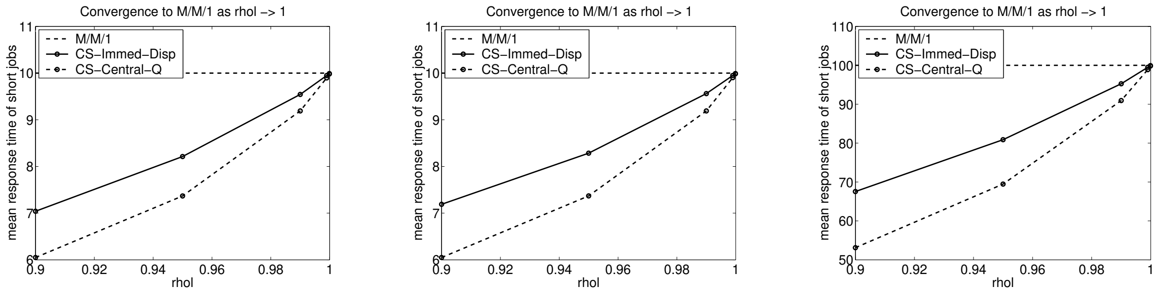
We performed event-driven simulations of our cycle stealing algorithm in C on a 700MHz Pentium III processor with 256 MB RAM. We experimented with a range of loads (ρ_S, ρ_L), various definitions of short and long, and different job size distributions (exponential and Coxian with $C^2 = 8$). Each experiment 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.

In Figure 10 we show just a *small subset* of our experiments, restricted to exponential job sizes, where ρ_S is held fixed at 0.9 and ρ_L is allowed to range over all stable values. Almost all of our simulation results were within 1-2 percent of predicted analysis; in some cases the simulation numbers were a little higher than analysis and in some cases a little lower. Simulation replications were quite consistent at low loads, but 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. We do not show the Coxian plots, but we do include those results in our discussion below.

Over all our simulation experiments, we found that with respect to mean response time of *long jobs*, there is no visible discrepancy between analysis and simulation under either cycle stealing algorithm. To understand why this is the case for `CS-Immediate-Dispatch`, recall that under `CS-Immediate-Dispatch` there are two Markov chains, one for the long host and one for the short. The Markov chain representing the state at the long host is exact (it involves no busy period approximations). Thus it is understandable that there is no discrepancy for the long jobs. Under `CS-Central-Queue` there is a single Markov chain, which does involve busy period approximations. The busy periods are up-down arrows in this chain. Nonetheless, recall that the performance of the long jobs, as described in Section 4.3 depends only on the parameters a_1 and a_2, which are the probabilities of being in Region 1 versus Region 2. These stationary probabilities are averages, which are likely less sensitive to the very high moments in the lengths of the busy periods.

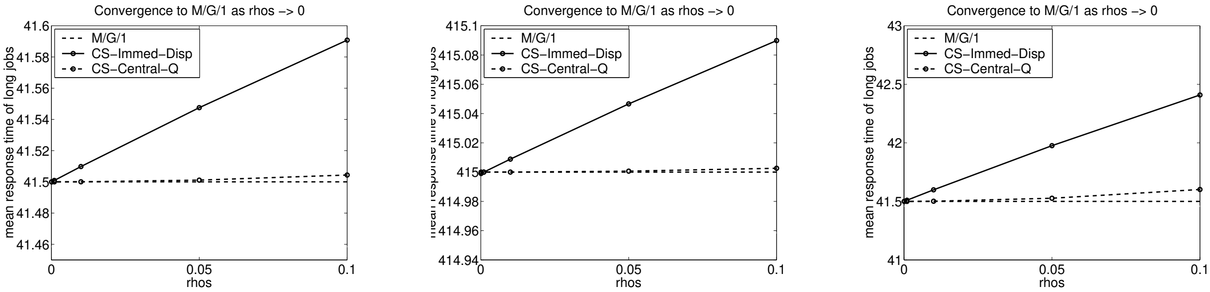
With respect to mean response time of *short jobs*, there was more discrepancy between analysis and simulation. This discrepancy was limited to the case of high load and was more prevalent in the case where long jobs were very long. It is easy to see why short jobs are more greatly impacted by the long busy period approximations under both cycle stealing algorithms: The length of the busy period of the long jobs affects the left/right motion of the chain which represents the number of small jobs. Since we only match the length of the busy period to three moments, we may be creating error in the number of short jobs. Furthermore, observe that since these busy periods consist primarily of long jobs, the variability in the length of these busy periods becomes more pronounced when long 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

Validation against Limiting Case 1



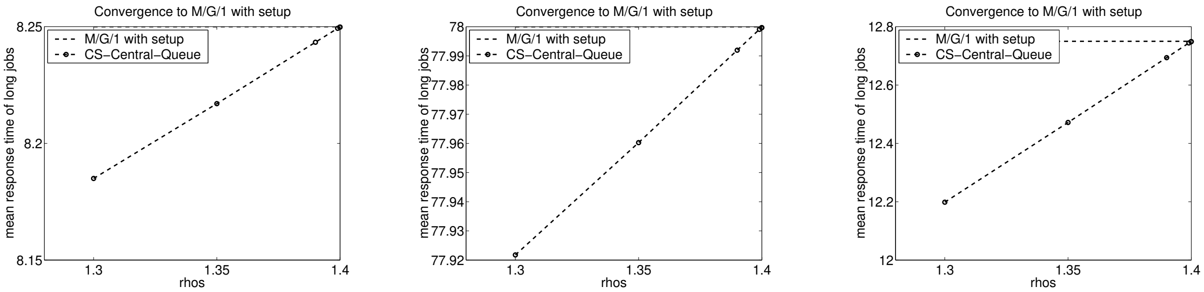
(a) shorts 1, longs 1 (b) shorts 1, longs 10 (c) shorts 10, longs 1

Validation against Limiting Case 2



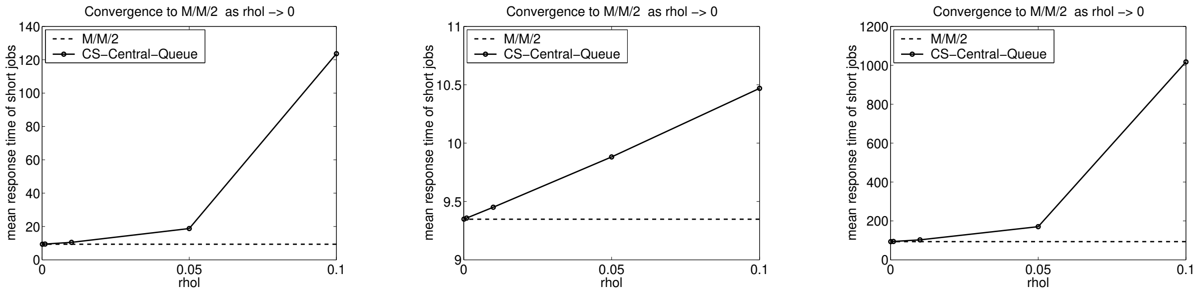
(a) shorts 1, longs 1 (b) shorts 1, longs 10 (c) shorts 10, longs 1

Validation against Limiting Case 3



(a) shorts 1, longs 1 (b) shorts 1, longs 10 (c) shorts 10, longs 1

Validation against Limiting Case 4



(a) shorts 1, longs 1 (b) shorts 1, longs 10 (c) shorts 10, longs 1

Figure 9: *Validation of analysis against four limiting cases. In row 1 $\rho_S = 0.9$. As $\rho_L \to 1$, response times for short jobs under both cycle stealing algorithms converge to an M/M/1 with load ρ_S. In row 2 $\rho_L = 0.9$. As $\rho_S \to 0$, response times for long jobs under both cycle stealing algorithms converge to an M/G/1/ with load ρ_L. In row 3 $\rho_L = 0.6$. As $\rho_S \to 1.4$, under* `CS-Central-Queue`*, response times of long jobs converge to an M/G/1 with setup $S \sim Exp(2\mu_S)$. In row 4 $\rho_S = 1.89$. As $\rho_L \to 0$, under* `CS-Central-Queue`*, short jobs converge to an M/M/2.*

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 never more than* 10%*, and that difference occurred rarely and only at high traffic intensity.*

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.

7 Results of Analysis

Recall that the motivation behind cycle-stealing algorithms is to improve the performance of "short" jobs without inflicting too much penalty on "long" jobs. Some penalty to long jobs is inevitable, of course, since our model is non-preemptive, meaning that a long arrival can't interrupt a short job running at its server.

In this section we will study the results of our analysis of the two cycle-stealing algorithms, `CS-Immediate-Dispatch` and `CS-Central-Queue`. All figures will be organized into two parts, where the first part will show the benefit to short jobs and the second part will show the penalty to long jobs. In order to evaluate these benefits/penalties we compare with the `Dedicated` algorithm which involves no cycle stealing.

In Figure 11 and 12, we hold ρ_L (the load at the long server) fixed at two representative values and consider the full range of ρ_S. Recall from Section 5, for `Dedicated` we can never have $\rho_S > 1$. However for `CS-Central-Queue`, ρ_S is allowed as high as $2 - \rho_L$. For `CS-Immediate-Dispatch`, ρ_S is allowed as high as some intermediate value shown in Figure 8, which is not as high as $2 - \rho_L$ and yet is higher than `Dedicated`.

Figure 11 shows analytical results in the case where both shorts and longs come from an exponential distribution. In row 1 and row 2 we fix $\rho_L = 0.5$ and vary ρ_S over all stable values. In row 3 and row 4 we fix $\rho_L = 0.8$ and vary ρ_S over all stable values.

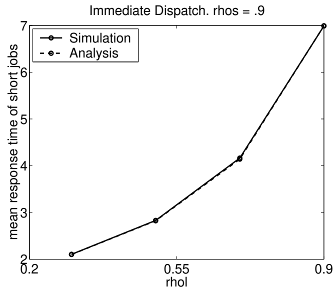
Looking at Figure 11 row 1, column (a) (where shorts and longs have mean size 1 and we fix $\rho_l = 0.5$), we see that the short jobs benefit tremendously from cycle stealing. For $\rho_S > 0.8$, the mean improvement of cycle stealing algorithms over `Dedicated` servers is over an order of magnitude. As $\rho_S \to 1$, the mean response time under `Dedicated` goes to infinity, whereas it is only around 4 under `CS-Immediate-Dispatch` and only around 3 under `CS-Central-Queue`. (Graphs have been truncated so as to fit on the page). This shows the huge benefit that short jobs obtain by being able to steal idle cycles from the long host.

Again looking at Figure 11 row 1, column (a) we see that the improvement of `CS-Central-Queue` over `CS-Immediate-Dispatch` is also vast. As $\rho_S \to 1.3$, the mean response time under `CS-Immediate-Dispatch` goes to infinity whereas it is only around 7 under `CS-Central-Queue`. This makes sense since under `CS-Immediate-Dispatch` only *new* short arrivals can benefit from idle cycles, whereas under `CS-Central-Queue` the long host, when seeing no long jobs, can take on short jobs already waiting.

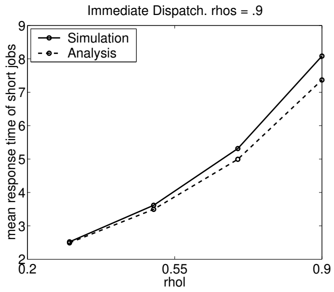
Looking at Figure 11 row 1, columns (b) and (c), we see that trends are similar to column (a), with only the absolute magnitude of the numbers growing.

Next we look at Figure 11 row 2, column (a) (where shorts and longs have mean size 1 and we fix $\rho_l = 0.5$) and find that the penalty imposed on long jobs by cycle stealing is relatively small. The penalty increases with ρ_S, but even when $\rho_S = 1$, the penalty to long jobs is only 20% under `CS-Central-Queue` and 50% under `CS-Immediate-Dispatch` (compared with the unbounded

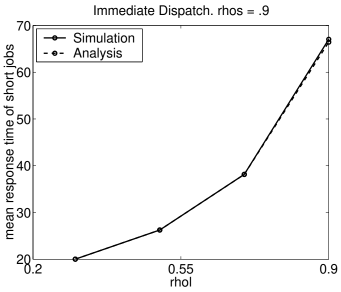
Validation against Simulation – Immediate Dispatch, Performance of Shorts



(a) shorts 1, longs 1

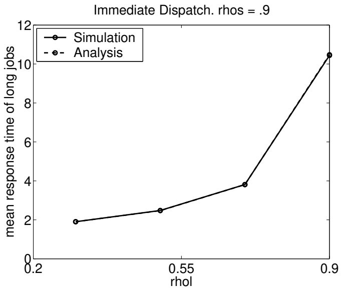


(b) shorts 1, longs 10

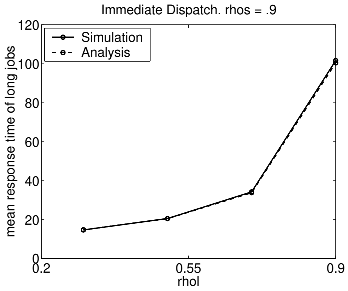


(c) shorts 10, longs 1

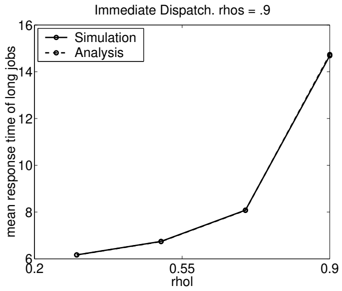
Validation against Simulation – Immediate Dispatch, Performance of Longs



(a) shorts 1, longs 1

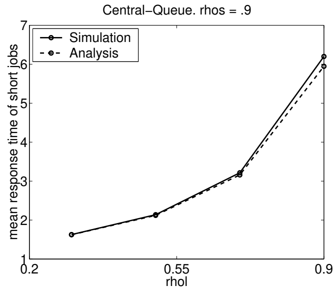


(b) shorts 1, longs 10

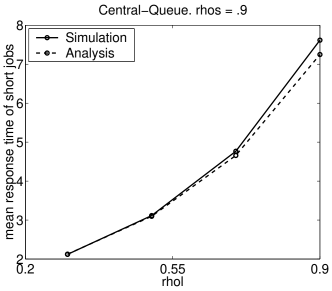


(c) shorts 10, longs 1

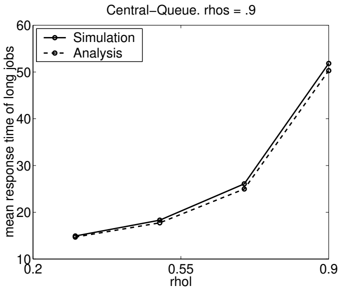
Validation against Simulation – Central-Queue, Performance of Shorts



(a) shorts 1, longs 1

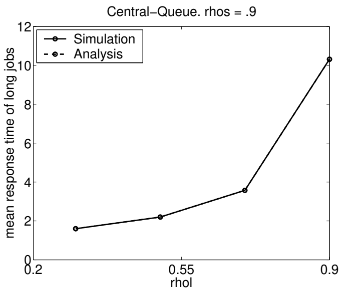


(b) shorts 1, longs 10

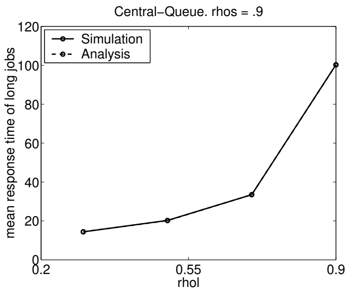


(c) shorts 10, longs 1

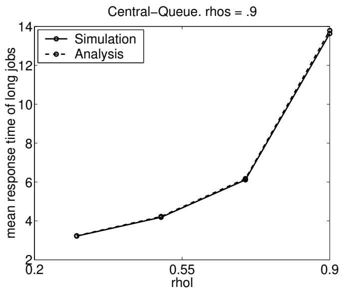
Validation against Simulation – Central-Queue, Performance of Longs



(a) shorts 1, longs 1



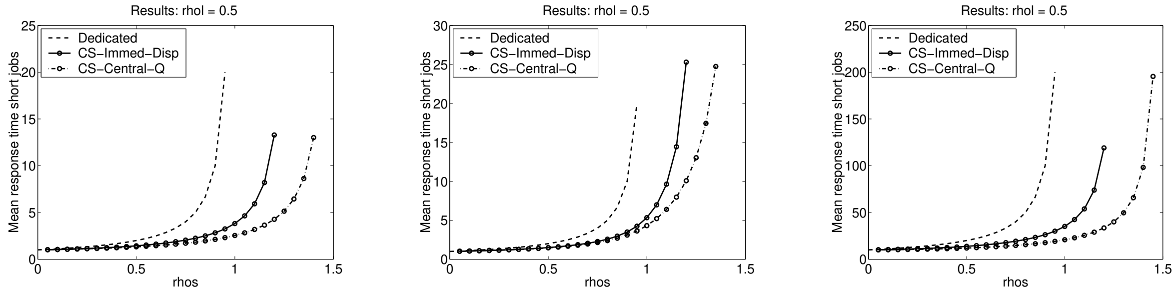
(b) shorts 1, longs 10



(c) shorts 10, longs 1

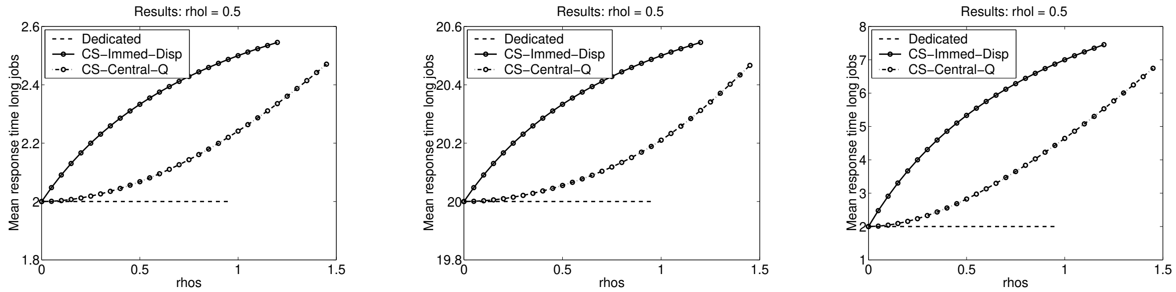
Figure 10: *Validation of Analysis against simulation. Row 1 shows performance of short jobs under* `CS-Immediate-Dispatch` *where we fix* $\rho_S = 0.9$ *and* ρ_L *varies. Row 2 shows performance of long jobs under* `CS-Immediate-Dispatch` *where we fix* $\rho_S = 0.9$ *and* ρ_L *varies. Row 3 shows performance of short jobs under* `CS-Central-Queue` *where* $\rho_S = 0.9$ *and* ρ_L *varies. Row 4 shows performance of long jobs under* `CS-Central-Queue` *where* $\rho_S = 0.9$ *and* ρ_L *varies.*

How shorts gain from cycle-stealing – $\rho_L = 0.5$



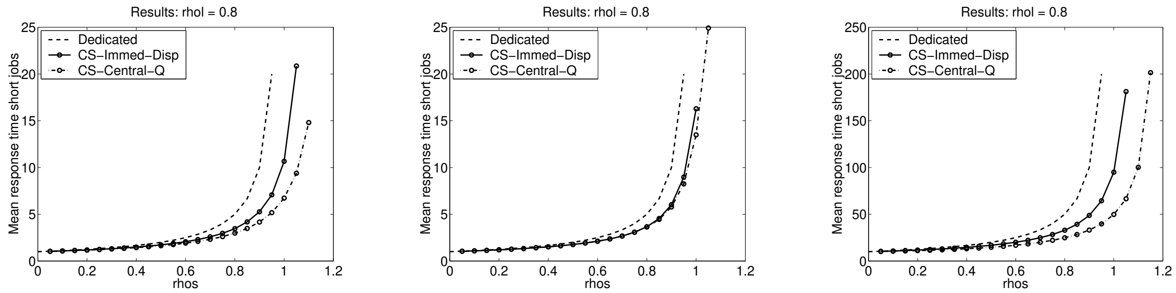
(a) shorts 1, longs 1 (b) shorts 1, longs 10 (c) shorts 10, longs 1

How longs suffer from cycle-stealing – $\rho_L = 0.5$



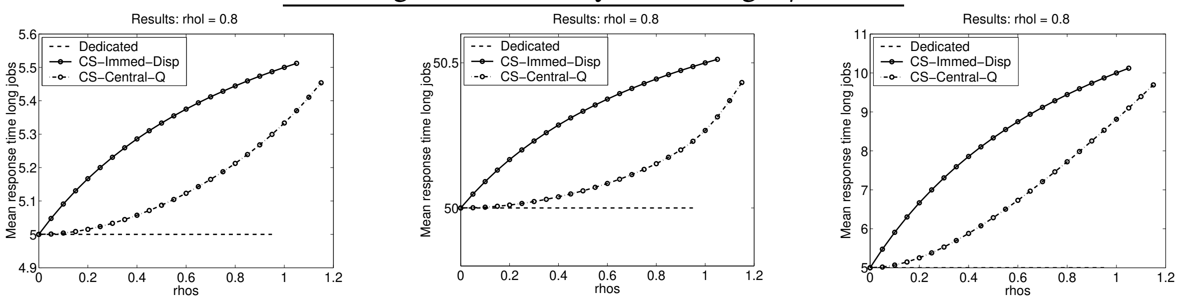
(a) shorts 1, longs 1 (b) shorts 1, longs 10 (c) shorts 10, longs 1

How shorts gain from cycle-stealing – $\rho_L = 0.8$



(a) shorts 1, longs 1 (b) shorts 1, longs 10 (c) shorts 10, longs 1

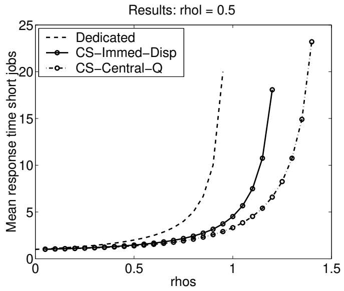
How longs suffer from cycle-stealing – $\rho_L = 0.8$



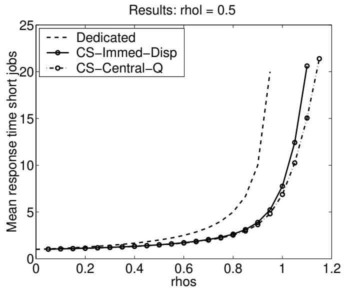
(a) shorts 1, longs 1 (b) shorts 1, longs 10 (c) shorts 10, longs 1

Figure 11: *Results of analysis, in the case where shorts and longs are drawn from exponential distributions.*

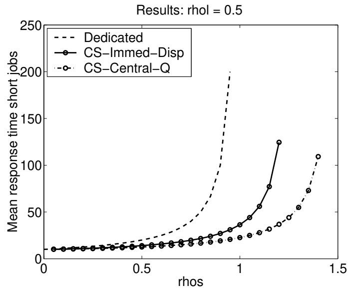
How shorts gain from cycle-stealing – $\rho_L = 0.5$



(a) shorts 1, longs 1

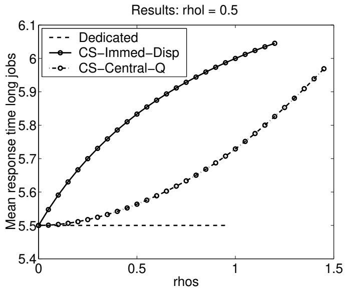


(b) shorts 1, longs 10

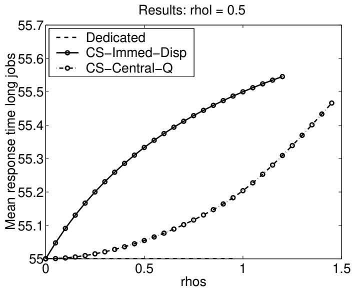


(c) shorts 10, longs 1

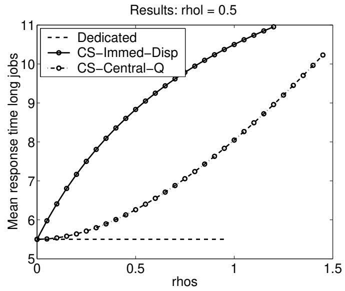
How longs suffer from cycle-stealing – $\rho_L = 0.5$



(a) shorts 1, longs 1

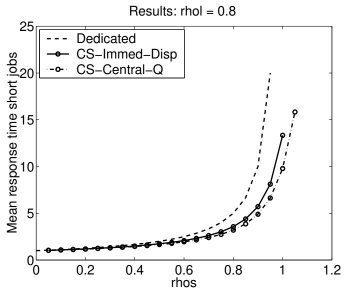


(b) shorts 1, longs 10

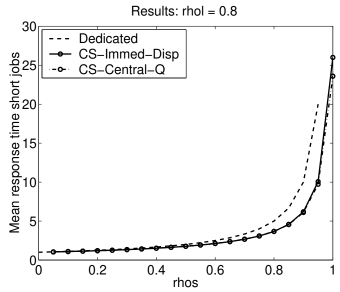


(c) shorts 10, longs 1

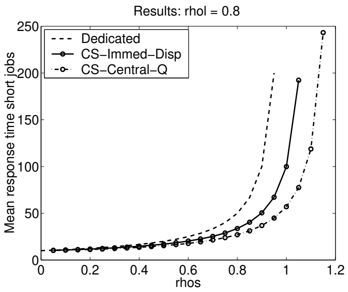
How shorts gain from cycle-stealing – $\rho_L = 0.8$



(a) shorts 1, longs 1

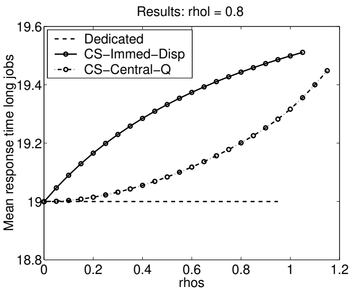


(b) shorts 1, longs 10

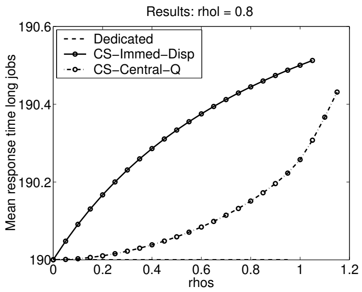


(c) shorts 10, longs 1

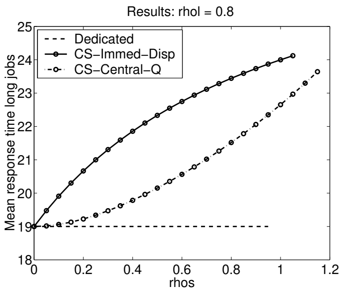
How longs suffer from cycle-stealing – $\rho_L = 0.8$



(a) shorts 1, longs 1



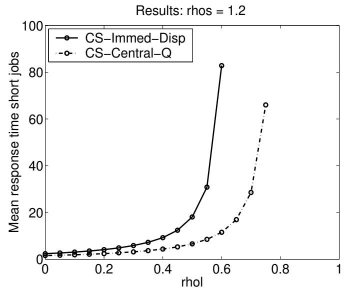
(b) shorts 1, longs 10



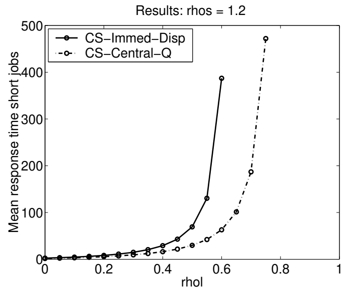
(c) shorts 10, longs 1

Figure 12: *Results of analysis, in the case where longs are drawn from Coxian distribution with appropriate mean and $C^2 = 8$. Response times are shown as a function of ρ_S.*

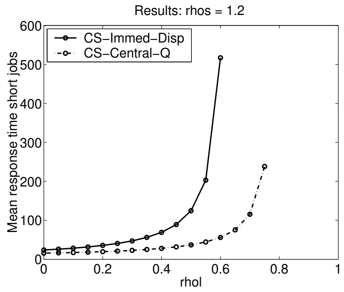
How shorts gain from cycle-stealing – $\rho_S = 1.2$



(a) shorts 1, longs 1

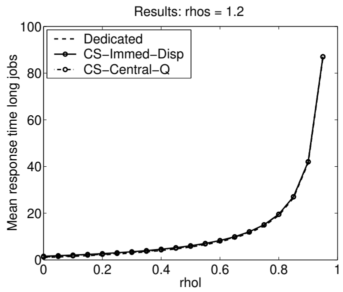


(b) shorts 1, longs 10

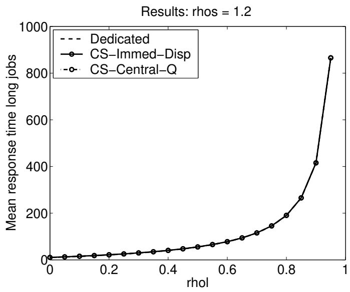


(c) shorts 10, longs 1

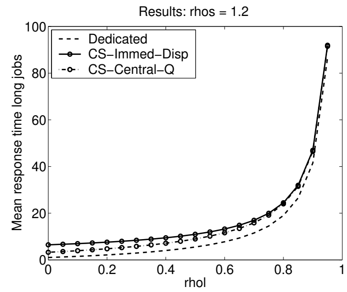
How longs suffer from cycle-stealing – $\rho_S = 1.2$



(a) shorts 1, longs 1

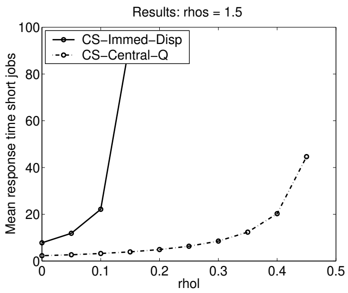


(b) shorts 1, longs 10

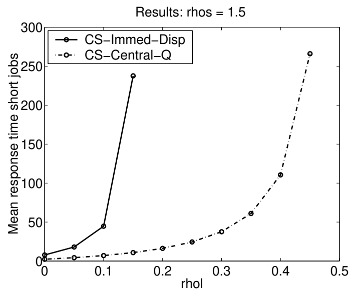


(c) shorts 10, longs 1

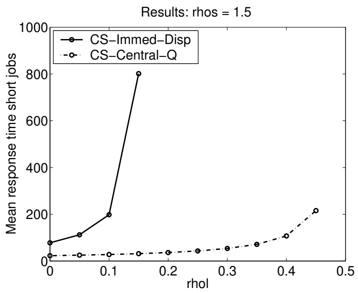
How shorts gain from cycle-stealing – $\rho_S = 1.5$



(a) shorts 1, longs 1

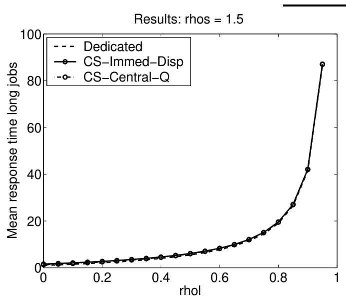


(b) shorts 1, longs 10

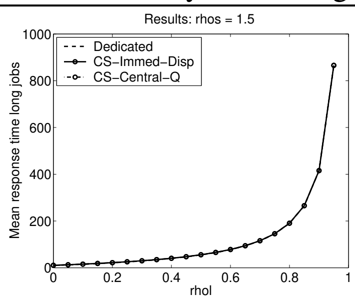


(c) shorts 10, longs 1

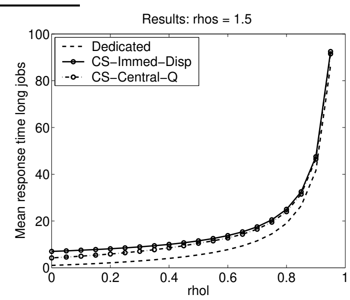
How longs suffer from cycle-stealing – $\rho_S = 1.5$



(a) shorts 1, longs 1



(b) shorts 1, longs 10



(c) shorts 10, longs 1

Figure 13: *Results of analysis, in the case where longs are drawn from Coxian distribution with appropriate mean and $C^2 = 8$. Response times are shown as a function of ρ_L.*

improvement for the shorts). For the case of column (b), where shorts are shorter than longs, this penalty drops to only 2% under `CS-Central-Queue` and 5% under `CS-Immediate-Dispatch`. For the case of column (c), where shorts are longer than longs, the penalty is greater. This is to be expected since jobs are not preemptible and a long job may now get stuck waiting behind a short job ten times its size. It's important to observe, however, that for all values of ρ_S, the penalty to long jobs is low compared with the tremendous performance improvement available to short jobs.

One interesting observation is that the penalty to long jobs appears lower under `CS-Central-Queue` than under `CS-Immediate-Dispatch`. At first this seems quite contrary, since under `CS-Central-Queue` it seems that more idle time is given to short jobs, thus the long jobs should suffer more. The reason this is not true is that under `CS-Central-Queue` the servers are renamable. This means that a long job arriving to find both servers serving short jobs need only wait for *the first* of the two servers to free up under `CS-Central-Queue`.

Considering Figure 11 rows 3 and 4, where we now fix $\rho_L = 0.8$, we see that results are largely similar to the case $\rho_L = 0.5$, except that the benefits to short jobs are not quite as great and the penalty to long jobs is also not as high. This is to be expected since now there are fewer idle cycles to steal. Still the performance improvement of cycle stealing over `Dedicated` servers is orders of magnitude for higher ρ_S.

Figure 12 is the counterpart to Figure 11, where now long jobs are drawn from a Coxian distribution with the appropriate mean and $C^2 = 8$, representing higher variability in the long jobs (short jobs are still drawn from an exponential distribution). Increasing the variability of the long job service time does not seem to have much effect on the mean benefit that cycle stealing offers to short jobs. With respect to the impact on long jobs, the long jobs end up with higher overall response times due to their higher variability, but similar absolute increase. The impact of cycle stealing on the long jobs is therefore considerably lessened when the variability in long job service times is increased. In fact, even for $\rho_S = 1$, under the case where shorts are shorter than longs (case (b)), the penalty to long jobs is less than 1% under both cycle stealing algorithms. For the case where shorts are indistinguishable from longs (case (a)), the penalty to longs is still under 10% for `CS-Immediate-Dispatch` and under 5% for `CS-Central-Queue`.

Until now we have not considered the case where ρ_{hL}, the load at the long host, is close to 1. It is interesting to ask what the penalty of cycle stealing to long jobs looks like as ρ_{hL} gets closer to 1. To investigate this question, we again consider the setup in Figure 12, except that this time we look at response time as a function of ρ_L, where we fix $\rho_S = 1.2$ and later $\rho_S = 1.5$ and consider the full range of ρ_L, as shown in Figure 13.

To understand these plots, it helps to first recall that Figure 7 limits the range of ρ_L under which `CS-Immediate-Dispatch` and `CS-Central-Queue` are stable. Specifically, Figure 7 shows that when $\rho_S = 1.2$, `CS-Immediate-Dispatch` is only stable for $\rho_L < .65$ (approximately) and `CS-Central-Queue` is only stable for $\rho_L < 0.8$ (approximately). For $\rho_S = 1.5$, the stability criteria on ρ_L are even more stringent, particularly for `CS-Immediate-Dispatch`. Figure 13 row 1 shows the mean response time for the short jobs under the two cycle stealing algorithms as a function of ρ_L, when we fix $\rho_S = 1.2$. Observe that as each algorithm nears its stability asymptote with respect to ρ_L, the response time shoots up to infinity (all graphs have been truncated). Thus, because `CS-Central-Queue` has a bigger stability region, its performance also appears far superior to `CS-Immediate-Dispatch`. Observe that we couldn't show the performance of `Dedicated` because it is unstable over the entire region.

Figure 13 row 2 considers the performance of the long jobs as a function of ρ_L where we again fix $\rho_S = 1.2$. Observe that the prior stability criterion on ρ_L was only based on keeping the *short* host stable. The long host, on the other hand, is stable for all values of ρ_L under `Dedicated` and under both cycle stealing algorithms. Figure 13 row 2 shows that cycle stealing does not penalize the long jobs, except in the case where the short jobs are much longer than the long jobs. In this case, cycle stealing penalizes the long jobs for lower loads; this penalty goes away for higher loads, since the short jobs can't get in to steal. Results for $\rho_S = 1.5$ are similar in trend.

In summary, we have seen that short jobs are tremendously helped by cycle stealing, and that `CS-Central-Queue` offers greater improvements to short jobs over `CS-Immediate-Dispatch`. We have also seen that, provided that short jobs are no longer than long jobs, the impact of cycle stealing to long jobs is negligible. Even when the short jobs are actually longer than the long jobs, the penalty to the long jobs is less, proportionally, than the benefit to the shorts. Furthermore, this impact is greater under `CS-Immediate-Dispatch` than under `CS-Central-Queue`. Thus `CS-Central-Queue` is always a better strategy than `CS-Immediate-Dispatch`, and both are far better than `Dedicated`.

8 Conclusion

The purpose of this paper is to analytically derive the benefit of cycle stealing where jobs normally destined for one machine may steal the idle cycles of another machine (the "donor" machine, a.k.a. "long" host). The motivation is that the jobs doing the stealing (the "beneficiaries, a.k.a. "short" jobs) will benefit immensely, while the donor jobs experience very little penalty, since (primarily) only their idle cycles are stolen. The paper considers two algorithms for cycle stealing: Immediate-Dispatch – where only newly arriving jobs can steal idle cycles – and Central-Queue – where the beneficiaries include both newly arriving jobs and already queued jobs.

At the onset of the paper we assumed that arriving jobs had been designated as being either "short" or "long", where "short" jobs were the ones permitted to do the stealing. However throughout the paper we also evaluate the case where "short" and "long" jobs are indistinguishable – a perhaps more applicable case – as well as the pathological case where "shorts" are longer than "longs."

Our results show that the beneficiaries (the shorts) can benefit by an order of magnitude under both cycle stealing algorithms. The donors (the longs) are penalized only by a small percentage, so long as shorts aren't longer than longs on average. Even when the short jobs are actually longer than the long jobs, the short jobs benefit more than the long jobs are penalized. Our results also show that `CS-Central-Queue` is a superior strategy to `CS-Immediate-Dispatch` both from the perspective of the benefit to short jobs and from the perspective of the impact on long jobs.

This paper presents the first analysis of cycle stealing. The analysis is an approximation, since it depends on approximating a busy period by a finite number of moments. The analysis can be made as precise as desired by using more moments – in this paper we use three. Still, even with just three moments, the analysis agrees well with simulation results. Furthermore, whereas generating a plot of simulation results requires an hour, generating a plot of analytical results requires only a couple seconds.

In this paper we make the assumption that jobs are not preemptible. It is interesting to compare our task assignment policy with other non-preemptive policies. A natural non-preemptive policy which comes to mind is `M/G/2/SJF`. Here it is assumed that there is a *central* queue where jobs are held at the dispatcher, and short jobs are always given preference over long jobs at both hosts. It turns out that from the perspective of both the short and long jobs, the `M/G/2/SJF` policy sometimes outperforms our cycle stealing algorithms and sometimes does worse, depending on conditions like ρ_S, ρ_L, and the job size distributions. On the plus side, `M/G/2/SJF` offers the short jobs two servers, where both servers prioritize in favor of shorts. On the negative size, because `M/G/2/SJF` does not offer a dedicated short server, the short jobs sometimes end up getting stuck behind two long jobs, one at each host. With respect to the long jobs, on the negative side, `M/G/2/SJF` penalizes long jobs at both servers, but on the positive side, long jobs may benefit in situations where ρ_S is low and two long jobs end up capturing both hosts.

A natural followup problem to this paper is the situation of two hosts, both of which help each other. That is, both hosts are donors *and* beneficiaries. This problem is open as of the time of this paper.

9 Appendix

This section includes the virtual waiting time analysis used to derive the Laplace transform of the waiting time for long jobs under `CS-Immediate-Dispatch`, denoted by $T_Q^{(L)}$. The response time for long jobs $T^{(L)}$ is defined as

$$T^{(L)} = T_Q^{(L)} + X_L.$$

Summary of Results

$$\begin{aligned}
\tilde{T}_Q^{(L)}(s) &= \frac{s + \lambda_S - \tilde{X}_S(s)\lambda_S}{s - \lambda_L + \tilde{X}_L(s)\lambda_L}\pi_0, \\
E[T_Q^{(L)}] &= \frac{\rho_L}{1-\rho_L}\frac{E[X_L^2]}{2E[X_L]} + \frac{\rho_S}{1+\rho_S}\frac{E[X_S^2]}{2E[X_S]}, \\
E[(T_Q^{(L)})^2] &= \frac{\rho_L}{1-\rho_L}\frac{E[X_L^3]}{3E[X_L]} + \frac{\rho_S}{1+\rho_S}\frac{E[X_S^3]}{3E[X_S]} + \frac{\rho_L}{1-\rho_L}\frac{E[X_L^2]}{E[X_L]}\left(\frac{\rho_L}{1-\rho_L}\frac{E[X_L^2]}{2E[X_L]} + \frac{\rho_L}{1-\rho_L}\frac{E[X_S^2]}{2E[X_S]}\right).
\end{aligned}$$

where

$$\pi_0 = \frac{1-\rho_L}{1+\rho_S},$$

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

Analysis of Long Host

Let $W(t)$ be the virtual waiting time at time t. That is, a job arrival at time t would wait $W(t)$ before we start processing the job. We follow the following steps to obtain the moments of $W = \lim_{t\to\infty} W(t)$:

1. Set up the differential equation for $\tilde{W}(t,s)$.
2. Let $t \to \infty$; then, $\frac{d\tilde{W}(t,s)}{dt} = 0$ ($\tilde{W}(s)$ is expressed by π_0).
3. Evaluate $\tilde{W}(s=0)$ to obtain π_0.
4. Differentiate $\tilde{W}(s)$ to obtain moments of W.

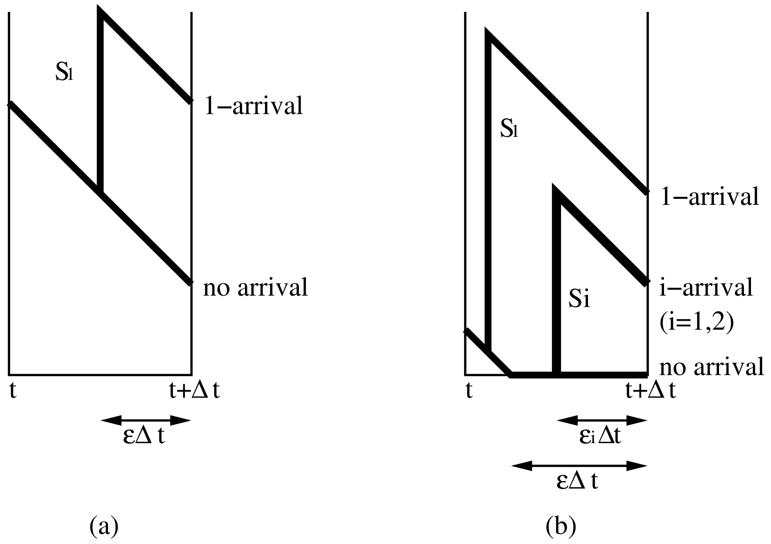


Figure 14: Virtual waiting time: relationship between $W(t)$ and $W(t+\Delta t)$

We first set up a differential equation. For this purpose, we carefully examine the relationship between $W(t)$ and $W(t+\Delta t)$.

If $W(t) \geq \Delta t$,

$$W(t+\Delta t) = \begin{cases} W(t) - \Delta t & \text{with probability } 1 - \lambda_L \Delta t, \\ W(t) + X_L - \Delta t & \text{with probability } \lambda_L \Delta t. \end{cases}$$

If $0 \leq W(t) < \Delta t$,

$$W(t+\Delta t) = \begin{cases} 0 & \text{with probability } 1 - \lambda_L \Delta t - \lambda_S \epsilon \Delta t, \\ W(t) + X_L - \epsilon_1 \Delta t & \text{with probability } \lambda_L \Delta t, \\ X_S - \epsilon_2 \Delta t & \text{with probability } \lambda_S \epsilon \Delta t. \end{cases}$$

Note that ϵ_1 and ϵ_2 are random variables with $0 \leq \epsilon_1, \epsilon_2 \leq 1$.

$$\begin{aligned}
&\tilde{W}(t+\Delta t, s) \\
&\quad \equiv \quad E[e^{-sW(t+\Delta t)}] \\
&\quad = \quad \int_{x=0}^{\infty} E[e^{-sW(t+\Delta t)} | W(t) = x] d\Pr(W(t) \leq x), \\
&\quad = \quad \int_{x=\Delta t}^{\infty} \left(e^{-s(x-\Delta t)} (1 - \lambda_L \Delta t) + E[e^{-s(x-\Delta t + X_L)}] \lambda_L \Delta t \right) d\Pr(W(t) \leq x) \\
&\qquad\quad + \int_{x=0^+}^{\Delta t} \left((1 - \lambda_L \Delta t - \lambda_S \epsilon \Delta t) + E[e^{-s(x + X_L - \epsilon_1 \Delta t)}] \lambda_L \Delta t + E[e^{-s(X_S - \epsilon_2 \Delta t)}] \lambda_S \epsilon \Delta t \right) d\Pr(W(t) \leq x) \\
&\qquad\quad + \left((1 - \lambda_L \Delta t - \lambda_S \Delta t) + E[e^{-s(x + X_L - \epsilon_1 \Delta t)}] \lambda_L \Delta t + E[e^{-s(X_S - \epsilon_2 \Delta t)}] \lambda_S \Delta t \right) \Pr(W(t) = 0), \\
&\quad = \quad \int_{x=0}^{\infty} \left(1 + (s - \lambda_L + \lambda_L \tilde{X}_L(s)) \Delta t + o(\Delta t) \right) e^{-sx} d\Pr(W(t) \leq x) \\
&\qquad\quad + \int_{x=0^+}^{\Delta t} \{ 1 - (\lambda_L + \lambda_S \epsilon) \Delta t + E[e^{-sx} e^{-sX_L} (1 + O(\Delta t))] \lambda_L \Delta t + E[e^{-sX_S} (1 + O(\Delta t))] \lambda_S \epsilon \Delta t
\end{aligned}$$

$$
\begin{aligned}
&\quad - \Big(1 + (s - \lambda_L + \lambda_L \tilde{X}_L(s))\Delta t + o(\Delta t)\Big) e^{-sx}\Big\} d\mathrm{Pr}(W(t) \le x) \\
&+ \big\{(1 - \lambda_L \Delta t - \lambda_S \Delta t) + E[e^{-sX_L}(1 + O(\Delta t))]\lambda_L \Delta t + E[e^{-sX_S}(1 + O(\Delta t))]\lambda_S \Delta t \\
&\quad - \Big(1 + (s - \lambda_L + \lambda_L \tilde{X}_L(s))\Delta t + o(\Delta t)\Big)\Big\} \mathrm{Pr}(W(t) = 0), \\
= \;& \Big(1 + (s - \lambda_L + \lambda_L \tilde{X}_L(s))\Delta t\Big) \tilde{W}(t, s) + \int_{x=0^+}^{\Delta t} O(\Delta t) d\mathrm{Pr}(W(t) \le x) \\
&+ \Big(-\lambda_S + \tilde{X}_S(s)\lambda_S - s\Big) \Delta t \mathrm{Pr}(W(t) = 0) + o(\Delta t).
\end{aligned}
$$

$$
\begin{aligned}
\frac{\tilde{W}(t + \Delta t, s) - \tilde{W}(t, s)}{\Delta t} &= \Big(s - \lambda_L + \lambda_L \tilde{X}_L(s)\Big) \tilde{W}(t, s) + \int_{x=0^+}^{\Delta t} O(1) d\mathrm{Pr}(W(t) \le x) \\
&\quad + \Big(-\lambda_S + \tilde{X}_S(s)\lambda_S - s\Big) \mathrm{Pr}(W(t) = 0) + \frac{o(\Delta t)}{\Delta t}.
\end{aligned}
$$

Letting $\Delta t \to 0$ in the above formula,

$$
\frac{d\tilde{W}(t, s)}{dt} = \Big(s - \lambda_L + \lambda_L \tilde{X}_L(s)\Big) \tilde{W}(t, s) + \Big(-\lambda_S + \tilde{X}_S(s)\lambda_S - s\Big) \mathrm{Pr}(W(t) = 0).
$$

Let $t \to \infty$. Then, $\frac{d\tilde{W}(t,s)}{dt} \to 0$.

$$
\begin{aligned}
\Big(s - \lambda_L + \tilde{X}_L(s)\lambda_L\Big) \tilde{W}(s) &= \Big(s + \lambda_S - \tilde{X}_S(s)\lambda_S\Big) \mathrm{Pr}(W(t) = 0), \\
\tilde{W}(s) &= \frac{s + \lambda_S - \tilde{X}_S(s)\lambda_S}{s - \lambda_L + \tilde{X}_L(s)\lambda_L} \pi_0,
\end{aligned}
$$

where $\pi_0 = \mathrm{Pr}(W(t) = 0)$ and $\lim_{t\to\infty} \tilde{W}(t, s) = \tilde{W}(s)$. Next, we will obtain π_0 by evaluating $W(s)$ at $s = 0$:

$$
\begin{aligned}
1 &= \tilde{W}(0),, \\
&= \left.\frac{1 - \frac{d\tilde{X}_S(s)}{ds}\lambda_S}{1 + \frac{d\tilde{X}_L(s)}{ds}\lambda_L}\right|_{s=0} \pi_0, \\
&= \frac{1 + E[X_S]\lambda_S}{1 - E[X_L]\lambda_L} \pi_0.
\end{aligned}
$$

The second equality is by L'Hopital's rule.

$$
\pi_0 = \frac{1 - \lambda_L E[X_L]}{1 + \lambda_S E[X_S]}.
$$

We will differentiate $\tilde{W}(s)$ and evaluate at $s = 0$ to obtain the moments of W, which is the time in queue $T_Q^{(L)}$ for the large jobs. Let

$$
\begin{aligned}
h(s) &= s + \lambda_S - \lambda_S \tilde{X}_S(s). \\
g(s) &= s - \lambda_L + \lambda_L \tilde{X}_L(s). \\
f(s) &= \frac{h(s)}{g(s)}.
\end{aligned}
$$

Then,

$$\begin{aligned} f(0) &= \frac{1}{\pi_0}, \\ h(0) = g(0) &= 0. \end{aligned}$$

Since

$$\begin{aligned} f'(0) &= \frac{h''(0) - f(0)g''(0)}{2g'(0)}, \\ f''(0) &= \frac{h'''(0) - 3f'(0)g''(0) - f(0)g'''(0)}{3g'(0)}, \end{aligned}$$

and

$$\begin{aligned} h(s) &= s + \lambda_S - \lambda_S \tilde{X}_S(s), \\ h'(s) &= 1 - \lambda_S \tilde{X}'_S(s), \\ h''(s) &= -\lambda_S \tilde{X}''_S(s), \\ h'''(s) &= -\lambda_S \tilde{X}'''_S(s), \\ g(s) &= s - \lambda_L + \lambda_L \tilde{X}_L(s), \\ g'(s) &= 1 + \lambda_L \tilde{X}'_L(s), \\ g''(s) &= \lambda_L \tilde{X}''_L(s), \\ g'''(s) &= \lambda_L \tilde{X}'''_L(s), \\ h'(0) &= 1 + \rho_S, \\ h''(0) &= -\lambda_S E[X_S^2], \\ h'''(0) &= \lambda_S E[X_S^3], \\ g'(0) &= 1 - \rho_L, , \\ g''(0) &= \lambda_L E[X_L^2] \\ g'''(0) &= -\lambda_L E[X_L^3], \end{aligned}$$

we have

$$\begin{aligned} f'(0) &= \frac{-\lambda_S E[X_S^2] - \frac{1}{\pi_0}\lambda_L E[X_L^2]}{2(1-\rho_L)}, \\ f''(0) &= \frac{\lambda_S E[X_S^3] - 3\frac{-\lambda_S E[X_S^2] - \frac{1}{\pi_0}\lambda_L E[X_L^2]}{2(1-\rho_L)}\lambda_L E[X_L^2] + \frac{1}{\pi_0}\lambda_L E[X_L^3]}{3(1-\rho_L)}. \end{aligned}$$

$$\begin{aligned} E[T_Q^{(L)}] &= -f'(0)\pi_0, \\ &= \frac{\rho_L}{1-\rho_L}\frac{E[X_L^2]}{2E[X_L]} + \frac{\rho_S}{1+\rho_S}\frac{E[X_S^2]}{2E[X_S]}, \\ E[(T_Q^{(L)})^2] &= f''(0)\pi_0, \\ &= \frac{\rho_L}{1-\rho_L}\frac{E[X_L^3]}{3E[X_L]} + \frac{\rho_S}{1+\rho_S}\frac{E[X_S^3]}{3E[X_S]} + \frac{\rho_L}{1-\rho_L}\frac{E[X_L^2]}{E[X_L]}\left(\frac{\rho_L}{1-\rho_L}\frac{E[X_L^2]}{2E[X_L]} + \frac{\rho_L}{1-\rho_L}\frac{E[X_S^2]}{2E[X_S]}\right). \end{aligned}$$

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] T. Brisco. DNS support for load balancing. RFC 1794, USC/Information Sciences Institute. Available at ftp://ds.internic.net/rfc/rfc1794.txt., April 1995.

[7] D.M.Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available Web server. In *Proceedings of the 41st IEEE Computer Society International Conference (COMPCON) '96*, pages 85–92, February 1996.

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

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

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

[20] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models*.

[21] 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.

[22] 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.

[23] 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.

[24] J. Sethuraman and M. S. Squillante. Optimal scheduling in multiclass multiserver systems. Technical report, IBM Research Division, 1998.

[25] M. S. Squillante, D. D. Yao, and L. Zhang. Web traffic modeling and web server performance analysis. Technical report, IBM Research Division, October 1998.

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

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

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

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