RC23098 (W0402-003) February 2, 2004
Mathematics

IBM Research Report

Analysis of Task Assignment with Cycle Stealing under Central Queue

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

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

Analysis of Task Assignment with Cycle Stealing under Central Queue

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

Abstract

We consider the problem of task assignment in a distributed server system, 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, where short and long jobs have generally-distributed service requirements, and arrivals are Poisson. We consider two variants of this problem: a central queue model and an immediate dispatch model. This paper presents the first analysis of cycle stealing under the central-queue model. (Cycle stealing under the immediate dispatch model is analyzed in [9]). The analysis uses a technique which we refer to as busy period transitions. Results show that cycle stealing can reduce mean response time for short jobs by orders of magnitude, while long jobs are only slightly penalized. Furthermore using a central queue yields significant performance improvement over immediate dispatch, both from the perspective of the benefit to short jobs and the penalty to long jobs.

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 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 hosts 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. We consider both the case where jobs are *immediately dispatched* 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.

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 high-volume Web sites [10, 20], studies of scalable systems for computers within an organization [18], and telecommunication systems with heterogeneous servers [4].

Previous work on Task Assignment

The analysis of task assignment policies has been the topic of many papers. We provide a brief overview, limiting our discussion to *non-preemptive* systems.

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 hosts, nor minimizes mean response time.

When the job processing requirements come from an exponential distribution, or one with increasing failure rate, the `M/G/k` policy has been proven to minimize mean response time and maximize utilization [24]. (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

[*] 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.

[†] Carnegie Mellon University, Graduate School of Industrial Administration. cuihong@cs.cmu.edu

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

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

[¶] IBM Thomas J. Watson Research Center, Mathematical Sciences Department. mss@watson.ibm.com

Name	Location	No. Hosts	Host Machine
Xolas [13]	MIT Lab for Computer Science	8	8-processor Ultra HPC 5000 SMP
Pleiades [12]	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 [6]. 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 [17]. Note: 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.*

the next job. 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 [7]. A related policy is the Shortest-Queue policy where incoming jobs are immediately dispatched to the host with the fewest number of jobs [23, 5].

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 analytically and empirically that the Dedicated policy far outperforms these other policies with respect to minimizing mean response time [8, 19]. In the Dedicated policy, some hosts are 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 Dedicated policy is 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, etc., and users must specify an estimated required service requirement for each job. The Dedicated policy performs well for the case of high-variability job size distributions because it isolates shorts jobs from the long jobs, as waiting behind the long jobs is very costly. The Dedicated policy is also popular in supermarkets and banks, where an "express" queue is created for "short" jobs.

Even when the job size is not known, 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 [7].

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: five consecutive short jobs may arrive, 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 we would like a policy which combines the variance-reducing benefit of the Dedicated policy with the high-utilization property of M/G/k and Shortest-Queue: We would segregate jobs by size to provide isolation for short jobs, but when the long job host is free, we would *steal* the long host's idle cycles to serve excess short jobs. This would both decrease the mean response time of short jobs, and enlarge the *stability region* of the overall system. Specifically, for systems where the short host is much 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 (see Section 3 for the stability regions for our cycle stealing algorithms). It is important, though, that we grant short jobs access to the long host only when that host is *free*, so we don't *starve* the long jobs, causing them undue delay. Nonetheless, because jobs are not preemptible, there will still be a penalty to a long job which arrives to find a short job serving at the long host.

Removing the distinction between short and long

Above we've used the terms "short host" to describe the host designated for "short" jobs and "long host" to describe the host designated for "long" jobs, but which can be used for short jobs when it is idle. Our reason for talking about a "short host" and a "long host" 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 situtation where there is a "beneficiary host" and a "donor host" where beneficiary jobs may use the "donor host" if it is idle. In particular, our analysis works equally well when short jobs are indistinguishable from long jobs — allowing us to simply derive the benefit that a donor host provides to a beneficiary host. 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. Even in this case, it will turn out that the penalty to "long" jobs is still dominated by the benefit to the "short" jobs.

Two cycle stealing algorithms

We propose two cycle stealing algorithms:

Cycle stealing with Immediate Dispatch (`CS-ID`): In this algorithm (shown in Figure 1(a)), 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 the long job host is not idle (either 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-ID` algorithm is an improvement over `Dedicated` for short jobs. However, only those short jobs arriving *after* the long host has entered an idle period can steal long cycles; 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-CQ`): In this algorithm (shown in Figure 1(b)), all jobs are held in a central queue. Whenever the short job host becomes idle, it picks the first short job in the queue to run. Whenever the long job host becomes idle, it picks the first long job in the queue. However, if there is no long job, the long host picks the first short job in the queue. A minor point: Whereas in `CS-ID` the short and long hosts are designated in advance, in `CS-CQ` 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-ID`, 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-CQ`.

Difficulty of analysis and new analytic approaches

Cycle stealing is a very old concept. Policies like `CS-ID` and `CS-CQ`, as well as others of a similar flavor, have been suggested in countless papers. However, the analysis of such policies has eluded researchers. Even for the simplest instance – where job arrivals are Poisson, and short and long jobs are drawn i.i.d. from different exponential distributions – the continuous-time Markov chain, while relatively easy to describe, is mathematically difficult. This is due to the fact that the state space

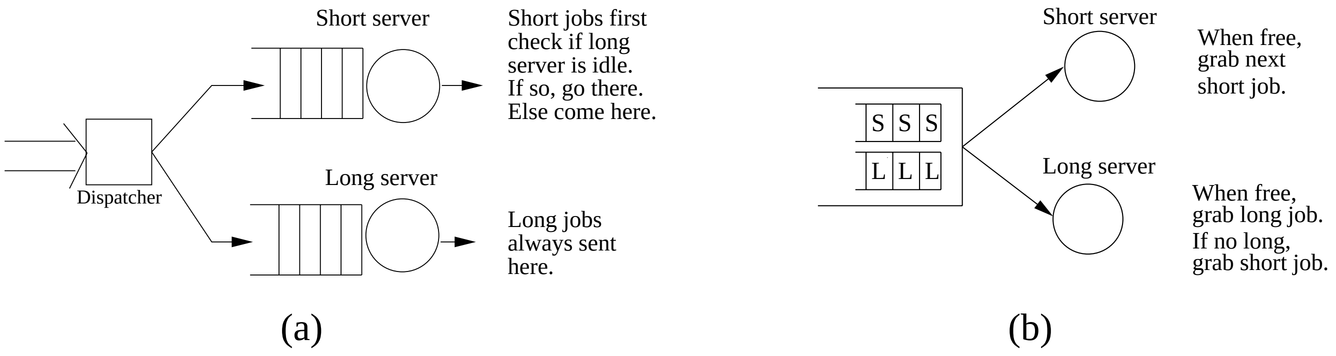
$$(number\ short\ jobs, number\ long\ jobs)$$

grows infinitely in two dimensions (2D).[1] 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 neither sufficiently accurate nor robust for our purposes.

In this paper we provide the first analysis of the `CS-CQ` policy. The analysis of `CS-ID` is given in [9]. In both cases, the analysis is approximate, but the approximation (based on moment matching) can be made as accurate as desired. We assume a Poisson arrival process for short and long jobs, which can be generalized to a MAP (Markovian Arrival Process) [11]. The service requirements of the short and long jobs are assumed to be drawn i.i.d. from any general distribution (which we approximate by a Coxian distribution).

The analysis of `CS-ID` hinges on decomposing the system into two separate stochastic processes. For `CS-CQ`, this technique does not work. Here we appear to need a Markov chain that is infinite in two dimensions, tracking both the number of short jobs and the number of long jobs, as the decision of which job the long host takes depends on both of these quantities. Our solution is to instead use a *single* chain which precisely tracks the number of short jobs, but where we use the transitions between the states to track the duration of various types of *busy periods* concerning the long jobs. We refer to these as *busy period transitions*. This yields a Markov chain infinite in only 1D. To understand how this works, observe that the effect of the long jobs on the short jobs may be captured by differentiating between three conditions: (i) there are zero long jobs, and shorts are being worked on by 2 servers; (ii) there is at least one long jobs and shorts are receiving no benefit, while they wait for the long host busy period to end; (iii) there is at least one long job, but a residual short job in service at the long host is blocking the long jobs, which are queueing and will create a busy period started by the sum of the sizes of all long jobs queued. We capture these busy period durations in our Markov chain transitions. We can derive any number of moments for these various busy periods. Our approach is to match the first three moments of each busy period, and verify that this provides sufficient accuracy via simulation (Section 4).

[1]The problem is equally hard when "short" and "long" jobs are indistinguishable with respect to size.



Short server
Short jobs first check if long server is idle. If so, go there. Else come here.
Dispatcher
Long server
Long jobs always sent here.
(a)
Short server
When free, grab next short job.
S S S
L L L
Long server
When free, grab long job. If no long, grab short job.
(b)

Figure 1: *(a) The* CS-ID *algorithm. (b) The* CS-CQ *algorithm.*

Notation	Definition
X_L	Size (service requirement) of long job.
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 $2\mu_S$.
N	Number of long jobs which arrive during E.
B_{N+1}	Busy period consisting of only long jobs, and started by work whose size is the sum of $N+1$ long jobs.

Table 2: *Notation necessary for analyzing* CS-CQ

2 Analysis of Cycle Stealing under Central Queue

2.1 Preliminary 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, $E[X_L^i]$). We will usually start by showing the case where job sizes are exponentially distributed, using 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 queues stable.

2.2 Formulating the Markov chain

In Figure 2(a) we show our chain representing CS-CQ. The first component of each state descriptor denotes the number of short jobs, which ranges from zero to infinity. The second component 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 component, 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 short job is assumed to be exponential, with rate μ_S, although this is straightforward to generalize using any phase-type (e.g., Coxian) distribution [15, 11].

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 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 while in region 2. This causes a transition into region 5. Before the long arrival can run, it must first wait for one of the short jobs to finish, since short jobs occupy both hosts. When one of the short jobs completes, a long job will move to occupy that host (that host will be renamed the long host) and we move to region 4. Thus, we remain in region 5 for time $E \sim Exp(2\mu_S)$. 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 work 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, short jobs can of course arrive and depart (being served with rate μ_S) . The only time short jobs are served at rate $2\mu_S$ is when there are zero long jobs in the system (region 2), or when a long job arrival finds that there are 2 short jobs being served (region 5). Notation is summarized in Table 2.

The chain in Figure 2(a) uses two types of busy period transitions: B_L and B_{N+1}. We can derive all the moments of these busy periods. Figure 2(b) is identical to Figure 2(a), except that the busy period transitions have been replaced by two-stage Coxian distributions, which allows

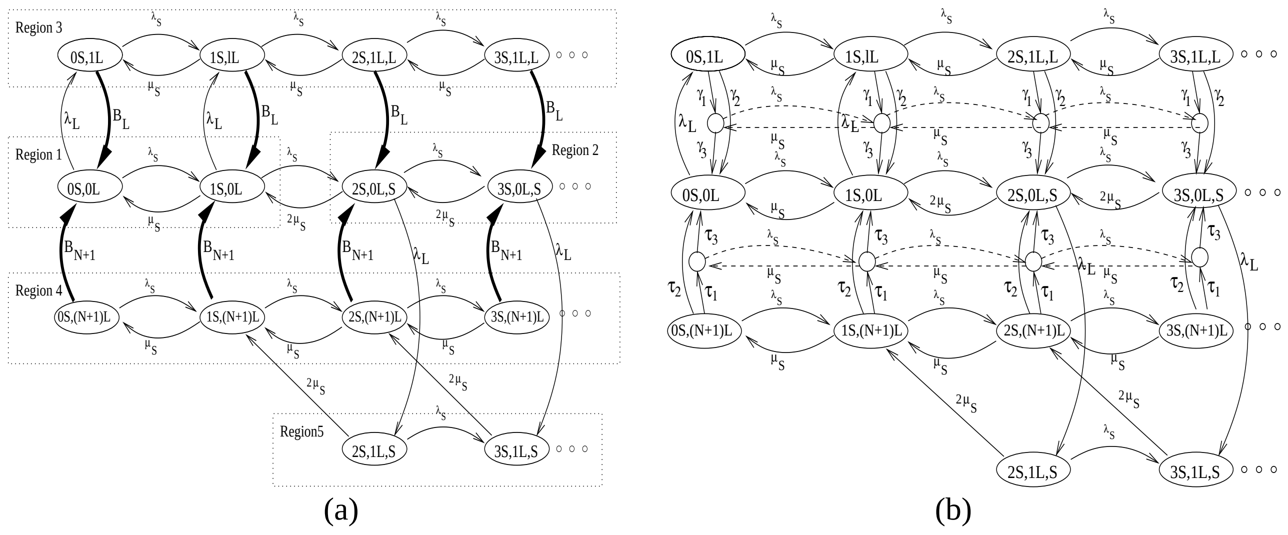


Figure 2: *(a) Chain corresponding to* `CS-CQ`. *The notation iS, jL represents i short jobs and j long jobs. The third field in the state denotes whether a short job or a long job is in service at the long host. Light arrows represent exponential rates. Bold arrows represent busy periods. B_L is a busy 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 period transitions have been replaced by Coxian distributions.*

us to model the first three moments of each busy period.[2] More moments could be modeled using a higher-degree Coxian, but three moments provide sufficient accuracy.

There is another slight approximation in our chain: It does not model any dependency between the time to move from region 5 to region 4 and the time to move from region 4 to regions 1 or 2. Simulation shows that this has negligible effect on mean response time.

2.3 Deriving the busy period transitions

In order to specify the Markov chain, we need to compute the first three moments of B_L and B_{N+1}.

B_L simply represents a busy period made up of only long jobs. Its Laplace transform is:

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

The moments of B_L can be obtained from the transform. B_{N+1} represents the length of a busy period made up of only long jobs, started by work whose size is the sum of $N+1$ long jobs, where N is the number of arrivals during $E \sim Exp(2\mu_S)$.

The Laplace transform of B_{N+1} is given by

$$\begin{aligned}\widetilde{B_{N+1}}(s) \quad = \quad & \widetilde{E}(\lambda(1 - \widetilde{X_L}(s + \lambda - \lambda\widetilde{B_L}(s)))) \cdot \\ & \widetilde{X_L}(s + \lambda_L - \lambda_L \widetilde{B_L}(s)).\end{aligned}$$

From this transform, we can derive B_{N+1}'s moments. The derivation is omitted, but is available at [9].

2.4 Analysis of short and long jobs

Response time for short jobs

Since our Markov chain tracks the exact number of short jobs, the mean response time for the short jobs can now be computed by solving the Markov chain and then applying Little's Law [14]. To solve the 1D-infinite Markov chain, we apply standard matrix-analytic method [15, 11]. This is a compact and efficient approach that allows one to solve quasi-birth-death processes (QBDs) which are infinite in one dimension, where the chain repeats itself after some point, as does Figure 2. The repeating portion is represented as powers of a matrix, R, which can be added, as one adds a geometric series, to produce a single matrix. Again see [9] for details. Every plot in this paper which uses this matrix-analytic analysis (to solve multiple instances with different parameter values) was produced within a couple of seconds using the Matlab 6 environment.

Response time for long jobs

Long jobs 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-

[2]We assume that a 2-stage Coxian suffices to match the first 3 moments of any distribution of interest. This is true for many distributions with higher variability. For exact conditions on the set of distributions whose first 3 moments can be matched by a 2-stage Coxian see [16]. Distributions with lower variability require a Coxian with more than 2 stages.

distributed with rate μ_S, the response time for long jobs is the response time for an M/G/1 queue with setup time S:

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

where $E \sim Exp(2\mu_S)$. When a more general phase-type distribution is used for X_S, then $S = \min(Excess(X_S), Excess(X_S))$ with probability a_2.

Observe that S is defined entirely by what the first job in a busy period sees – this job arrives in Region 1 or Region 2. The expected waiting time for an M/G/1 queue of long jobs with setup time S is known [21]:

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

We thus have:

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

3 Stability for CS-ID and CS-CQ

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-ID` we will see that the region of stability is much wider, and for `CS-CQ` wider still.

Theorem 1 *For* `CS-CQ`*, the stability condition for long jobs is* $\rho_L < 1$*, and the stability condition for short jobs is* $\rho_S < 2 - \rho_L$*. For* `CS-ID`*, the stability condition for long jobs is* $\rho_L < 1$*, and the stability condition for short jobs is the solution to* $\rho_L < \frac{1}{\rho_S} + 1 - \rho_S$*.*

The proof of the above theorem is available at [9]. The restriction on ρ_S for each of the three task assignment policies is shown in Figure 3. 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-ID` and close to 2 under `CS-CQ`.

4 Validation of analytical method

As we are proposing a new analytical scheme to arrive at near-exact calculations of waiting times, it is import that we validate the accuracy of our method, particularly our approximation of the length of a busy period by its first three moments. We validate our method in two ways:

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 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 short jobs). 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 for exponential service times.

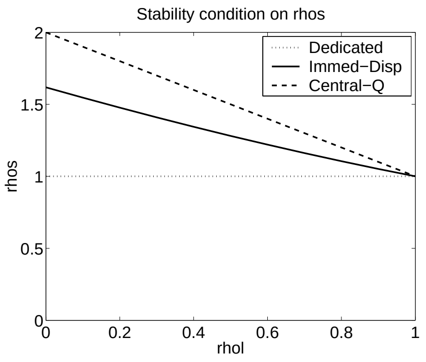


Figure 3: *Stability constraint on ρ_S for* `Dedicated`, `CS-ID`*, and* `CS-CQ`.

Validation against simulation: Having evaluated our approximation methods for limiting cases, we next use 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, 22]. Simulations were performed in C on a 700MHz Pentium III with 256 MB RAM.

All validation results are omitted for lack of space, but available in [9]. To summarize: 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 squared coefficient of variation $C^2 = 8$). The validation against known limiting cases was perfect. For the validation against simulation, we found that over all the simulation experiments, the difference between analysis and simulation was under 2% in almost all cases, and was never over 8%, and such a difference occurred rarely and only at very high load. It is also worth pointing out that for each results graph in [9], the simulation portion required *close to an hour* to generate, whereas the analysis portion required less than a second to compute.

5 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, though, since our model is non-preemptive. In this section we will study the results of our analysis of our cycle-stealing algorithms `CS-ID` and `CS-CQ`. All figures will be organized into two parts, where the first will show the benefit to short jobs and the second the penalty to long jobs. In order to evaluate these benefits/penalties we compare with the `Dedicated` algorithm.

In Figures 4 and 5, we hold ρ_L fixed at 0.5 and consider the full range of ρ_S. Recall from Section 3, for `Dedicated` we can never have $\rho_S > 1$. However for `CS-CQ`, ρ_S is allowed as high as $2 - \rho_L$, and for `CS-ID`, ρ_S is allowed as high as some intermediate value shown in Figure 3, which is not as high as $2 - \rho_L$ and yet is higher than `Dedicated`.

Figure 4 shows analytical results in the case where both shorts and longs come from an exponential distribution. Looking at row 1, column (a) (where shorts and longs have mean size 1), 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` is over an order of magnitude. As $\rho_S \rightarrow 1$, the mean response time under `Dedicated` goes to infinity, whereas it is 4 under `CS-ID` and 3 under `CS-CQ`. (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.

Still looking at Figure 4 row 1, column (a) we see that the improvement of `CS-CQ` over `CS-ID` is also vast. As $\rho_S \rightarrow 1.3$, the mean response time under `CS-ID` goes to infinity whereas it is approximately 7 under `CS-CQ`. This follows as under `CS-ID` only *new* short arrivals can benefit from idle cycles, whereas under `CS-CQ` waiting short jobs may benefit. Looking at Figure 4 row 1, columns (b) and (c), we see that trends are similar to column (a), with only the absolute magnitude of the numbers growing.

Figure 4 row 2, column (a) (where shorts and longs have mean size 1) shows 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 10% under `CS-CQ` and 25% under `CS-ID` (compared with the unbounded improvement for the short jobs). In column (b), where shorts are shorter than longs, this penalty drops to only 1% under `CS-CQ` and 2.5% under `CS-ID`. In column (c), where shorts are longer than longs, the penalty is greater. (This is to be expected since a long job may get stuck behind a "short" job ten times its size.) However, for all values of ρ_S, the penalty to long jobs is low compared with the improvement of short jobs.

One interesting observation is that the penalty to long jobs appears lower under `CS-CQ` than under `CS-ID`. At first this seems quite contrary, since under `CS-CQ` more idle time is given to short jobs, thus it is reasonable to expect the long jobs should suffer more. The reason this is not true is that under `CS-CQ` the servers are re-namable. Thus 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-CQ`.

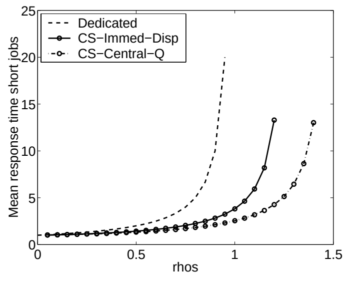
Other experiments, at higher values of ρ_L, show behavior largely similar to the case $\rho_L = 0.5$, except that both the benefits to short jobs and the penalty to long jobs are reduced, see [9]. This is to be expected since there are fewer idle cycles to steal. Nevertheless, the performance improvement of cycle stealing over `Dedicated` is still orders of magnitude for high ρ_S.

Figure 5 is the counterpart to Figure 4, where long jobs are drawn from a Coxian distribution with appropriate mean and $C^2 = 8$, representing higher variability in the long jobs (short jobs are still exponential). 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. The long jobs have higher overall response times due to their higher variability, but similar absolute increase. The percentage penalty of the long jobs is therefore considerably lessened when the variability in long job service times is increased. In fact, even for $\rho_S = 1$, in 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-ID` and under 5% for `CS-CQ`.

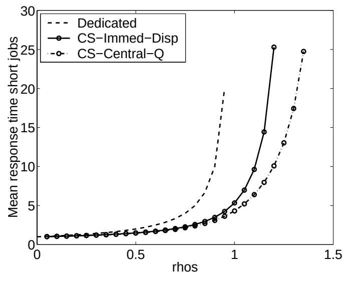
Until now we have not considered the case where ρ_{hL}, the load at the long host, is close to 1. To investigate this question, we again consider the setup in Figure 5, except that this time we look at response time as a function of ρ_L, fixing $\rho_S = 1.5$ as shown in Figure 6. To understand Figure 6, it helps to recall that Figure 3 shows the range of ρ_L under which `CS-ID` and `CS-CQ` are stable. Specifically, when $\rho_S = 1.5$, `CS-ID` is only stable for $\rho_L < .167$ and `CS-CQ` only for $\rho_L < 0.5$. Figure 6 row 1 shows the mean response time for short jobs under the two cycle stealing algorithms as a function of ρ_L. As each algorithm nears its stability asymptote, the response time rises to infinity (all graphs have been truncated). Thus, because `CS-CQ` has a larger stability region, its performance appears far superior to `CS-ID`. We couldn't show the performance of `Dedicated` because it is unstable over the entire region.

The prior stability criterion on ρ_L was only based on keeping the *short* host stable; the long host is stable for all values of ρ_L under `Dedicated` and both cycle stealing algorithms. Thus Figure 6 row 2 shows the performance of the long jobs as a function of ρ_L for `CS-CQ`, `CS-ID`, and `Dedicated`. Here we see that cycle stealing does not visibly 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, but

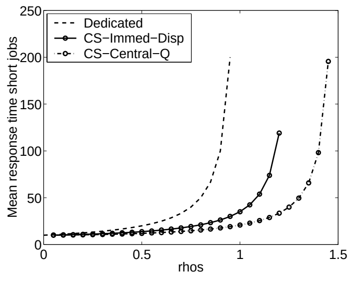
How shorts gain from cycle-stealing – $\rho_L = 0.5$



(a) shorts 1, longs 1

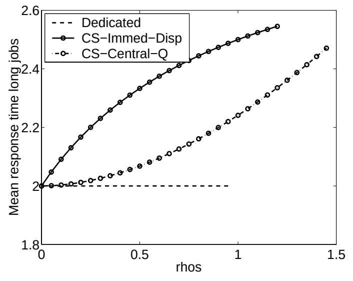


(b) shorts 1, longs 10

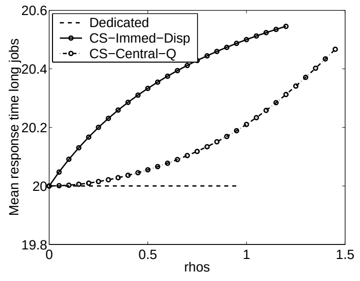


(c) shorts 10, longs 1

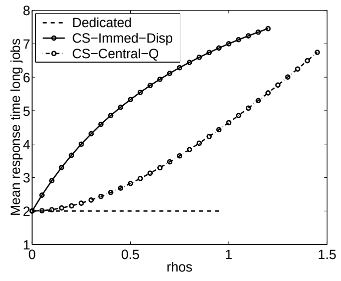
How longs suffer from cycle-stealing – $\rho_L = 0.5$



(a) shorts 1, longs 1



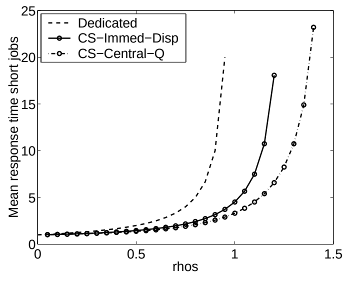
(b) shorts 1, longs 10



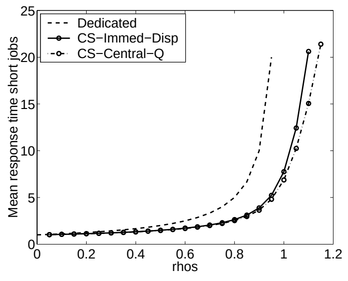
(c) shorts 10, longs 1

Figure 4: *Results of analysis, in the case where shorts and longs are drawn from exponential distributions.*

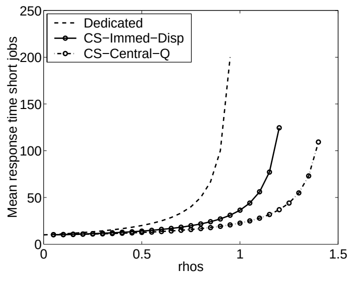
How shorts gain from cycle-stealing – $\rho_L = 0.5$



(a) shorts 1, longs 1

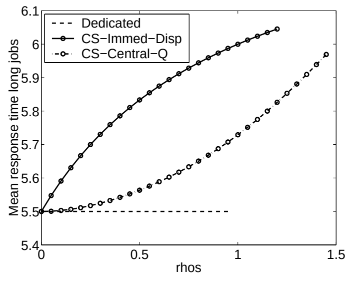


(b) shorts 1, longs 10

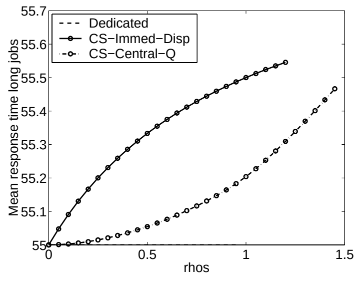


(c) shorts 10, longs 1

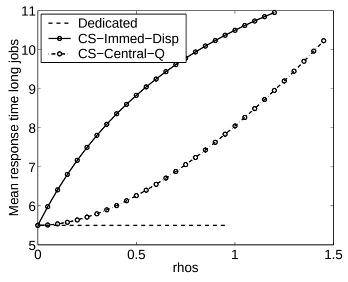
How longs suffer from cycle-stealing – $\rho_L = 0.5$



(a) shorts 1, longs 1



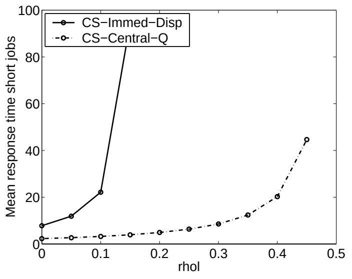
(b) shorts 1, longs 10



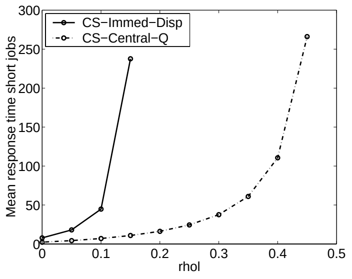
(c) shorts 10, longs 1

Figure 5: *Results of analysis, in the case where longs are drawn from Coxian distribution with appropriate mean and squared coefficient of variation $C^2 = 8$. Response times are shown as a function of ρ_S.*

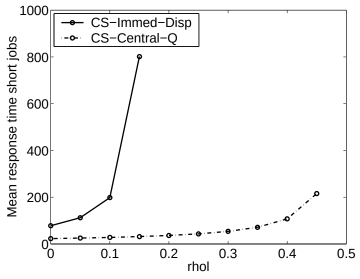
How shorts gain from cycle-stealing – $\rho_S = 1.5$



(a) shorts 1, longs 1

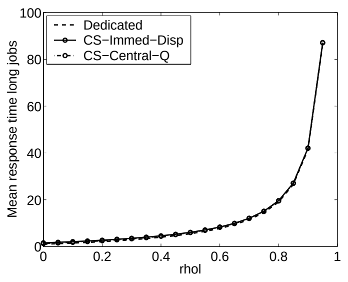


(b) shorts 1, longs 10

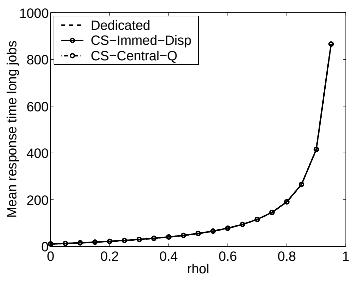


(c) shorts 10, longs 1

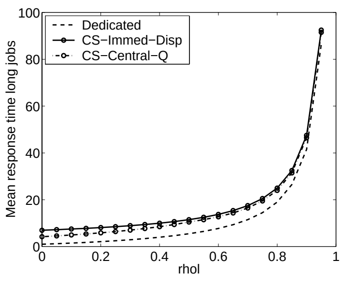
How longs suffer from cycle-stealing – $\rho_S = 1.5$



(a) shorts 1, longs 1



(b) shorts 1, longs 10



(c) shorts 10, longs 1

Figure 6: *Results of analysis, in the case where longs are drawn from Coxian distribution with appropriate mean and squared coefficient of variation $C^2 = 8$. Response times are shown as a function of ρ_L.*

the penalty vanishes for higher loads, since the short jobs can't get in to steal. Results for other values of ρ_S are similar in trend.

To summarize, we have seen that short jobs are tremendously helped by cycle stealing, and that `CS-CQ` offers greater improvements to short jobs than `CS-ID`. We have also seen that, provided that short jobs are no longer than long jobs, the impact of cycle stealing on long jobs is negligible. Even when the short jobs are longer than the long jobs, the penalty to the long jobs is less, proportionally, than the benefit to the shorts. This impact is greater under `CS-ID` than under `CS-CQ`. Thus `CS-CQ` is always superior to `CS-ID`, and both are far better than `Dedicated`.

6 Conclusion and Discussion

The purpose of this paper is to analytically derive the benefit of cycle stealing where jobs normally destined for one machine (the beneficiaries) may steal the idle cycles of another machine (the donor machine). The motivation is that the beneficiaries 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 were designated as either "short" or "long", where "short" jobs were 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 beneficiaries can benefit by an order of magnitude under both cycle stealing algorithms. The donors are penalized only by a small percentage, so long as they aren't shorter than the beneficiaries on average. Even when the beneficiaries are longer than the donors, the beneficiaries benefit more than the donors are penalized. We also find that `CS-CQ` is a superior strategy to `CS-ID` from the perspective of both the beneficiaries and the donors.

This paper presents the first analysis of cycle stealing under a Central Queue. The analysis involves creating a Markov chain that includes transitions that correspond to various types of busy periods. Our analysis is an approximation, since it depends on approximating these busy periods by a finite number of moments, but this approximation can be made as precise as desired by using more moments. Even with just three moments, our analysis agrees well with simulation. Furthermore, whereas generating a plot of simulation results typically requires an hour, generating the plot analytically requires only a couple seconds.

In this paper we make the assumption that jobs are not preemptible. If we allowed jobs to be preempted (interrupted) and subsequently resumed where they left off, the same basic approach would work, however, that is inconsistent with our model of supercomputing systems. We have also assumed homogeneous hosts. This assumption was simply made for ease of exposition. This work may be extended to hosts of different speeds.

It would be interesting to compare our task assignment policies with other non-preemptive policies. A natural candidate is `M/G/2/SJF`: A central queue holds jobs at the dispatcher, giving short jobs preference at both hosts. It turns out that from the perspective of both the short and long jobs, `M/G/2/SJF` sometimes outperforms our cycle stealing algorithms and sometimes does worse, depending on ρ_S, ρ_L, and the job size distributions. On the plus side, `M/G/2/SJF` offers the short jobs two servers where they have priority. But, because `M/G/2/SJF` does not offer a dedicated short server, the short jobs sometimes get 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 long jobs may benefit in situations where ρ_S is low and long jobs end up capturing both hosts.

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] 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.

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

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

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

[8] 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.

[9] M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, and M. Squillante. Analysis of task assignment with cycle stealing. Technical Report CMU-CS-02-158, Sept. 2002.

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

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

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

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

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

[15] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models:An Algorithmic Approach*. The Johns Hopkins University Press, 1981.

[16] T. Osogami and M. Harchol-Balter. Necessary and sufficient conditions for representing general distributions by Coxians. Technical Report CMU-CS-02-178, Computer Sciences Department, Carnegie Mellon, September 2002.

[17] 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.

[18] 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.

[19] 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.

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

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

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

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

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