An Adaptive Threshold-Based Policy for Sharing Servers with Affinities

Takayuki Osogami[1] Mor Harchol-Balter[2]
Alan Scheller-Wolf[3] Li Zhang[4]

February, 2004
CMU-CS-04-112

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We evaluate the performance of threshold-based job allocation policies in a heterogeneous distributed computing system, where servers may have different speeds, and jobs may have different service demands, importance, and/or affinities for different servers. We find that while threshold-based policies typically yield low mean response time, these policies are not robust with respect to fluctuations or misprediction of the load. We propose a new adaptive dual-threshold policy and show that this policy yields low mean response time while also being robust.

[1]Carnegie Mellon University, Computer Science Department, osogami@cs.cmu.edu
[2]Carnegie Mellon University, Computer Science Department, harchol@cs.cmu.edu. This work was supported by NSF Career Grant CCR-0133077, by NSF Grant CCR-0311383, NSF Grant-0313148, and by IBM via 2003 Pittsburgh Digital Greenhouse Grant.
[3]Carnegie Mellon University, GSIA, awolf@andrew.cmu.edu
[4]IBM Research, Thomas J. Watson Research Center, zhangli@us.ibm.com

Keywords: load sharing, threshold policies, affinities, adaptive, multiserver systems, dimensionality reduction, busy period transitions, $c\mu$ rule

1 Introduction

A common problem in distributed computing systems is the allocation of server time among queues of jobs (allocation policy). Specifically, there may be jobs originating at one (heavily loaded) server which are better off served by some other (more lightly loaded) server. Since estimating the system load is a difficult task, and in fact the system load often fluctuates over time, it is desirable that the allocation policy be robust against such mispredictions and fluctuations of the system load. Allocation policies are necessary, for example, when migrating processes in networks of workstations [10], when dynamically allocating resources in utility computing [1, 4, 5], and when assigning tasks in multiprocessor systems [22, 24].

The goal of this paper is to design and evaluate allocation policies for a general heterogeneous distributed computing system where (i) the jobs originating at different servers may have different mean size (processing requirement), (ii) the servers may have different speeds, (iii) jobs may have different affinities with different servers, i.e., a job may be processed more efficiently (have shorter duration) if run on one server than when run on another server, and (iv) different jobs may have different importance (weights). Our objective is twofold. First we seek to minimize the weighted average mean response time (weighted response time), $\sum_i c_i p_i E[R_i]$, where c_i is the weight (importance) of jobs originating at server i, λ_i is the average rate of jobs originating at server i (type i jobs), $p_i = \lambda_i / \sum_j \lambda_j$ is the fraction of type i jobs, and $E[R_i]$ is the mean response time[1] of type i jobs. Second, we want our policy to be robust against environmental changes, such as changes in load.

Figure 1 shows the model we consider for the case of two servers. Jobs arrive at queue 1 and queue 2 with average rates λ_1 and λ_2, respectively. Server 1 processes jobs in queue 1 with average rate μ_1 (jobs/sec), while server 2 can process jobs in queue 1 with average rate μ_{12} (jobs/sec) and can process jobs in queue 2 with average rate μ_2 (jobs/sec). We define $\rho_1 = \lambda_1/\mu_1$, $\rho_2 = \lambda_2/\mu_2$, and $\hat{\rho}_1 = \lambda_1/(\mu_1 + \mu_{12}(1-\rho_2))$. Here, $\hat{\rho}_1$ is the load of type 1 jobs assuming server 2 helps server 1 as much as possible, while processing all the type 2 jobs. Note that $\rho_2 < 1$ and $\hat{\rho}_1 < 1$ are necessary for the queues to be stable under any allocation policy, while $\rho_2 < 1$ and $\hat{\rho}_1 < 1$ are sufficient for the queues to be stable under any non-idling policy. Even in this simple model of just two servers, the optimal allocation policy is not known, despite the fact that this problem has been investigated in numerous papers [2, 13, 14, 29, 23, 21, 9, 11, 26, 27]. Below we will focus on two servers and two queues with Poisson arrivals and exponential job size distributions. Extensions to more general cases are discussed in Section 5.

One common allocation policy is based on the $c\mu$ rule [7], where a server processes jobs from the nonempty queue with the highest $c\mu$ value, biasing in favor of jobs with high c (high importance) and high μ (small expected size[2]). Under the $c\mu$ rule, server 2 in Figure 1 serves jobs from queue 1 (rather than queue 2) if $c_1\mu_{12} > c_2\mu_2$, or queue 2 is empty. The $c\mu$ rule is provably optimal in the limit as job size approaches zero (fluid model) [25]. However Squillante et. al. [23] as well as Harrison [13] have shown that $c\mu$ rule may lead to instability even if $\hat{\rho}_1 < 1$ and $\rho_2 < 1$. For example, the $c\mu$ rule may force server 2 to process jobs from queue 1 even when many jobs are built up at queue 2, leading to instability in queue 2 and under-utilization of

[1] Here response time refers to the total time from when a job is requested until the job is completed – this includes queueing time and service time.

[2] Note that the average size of a job is $1/\mu$, where μ is the average service rate.

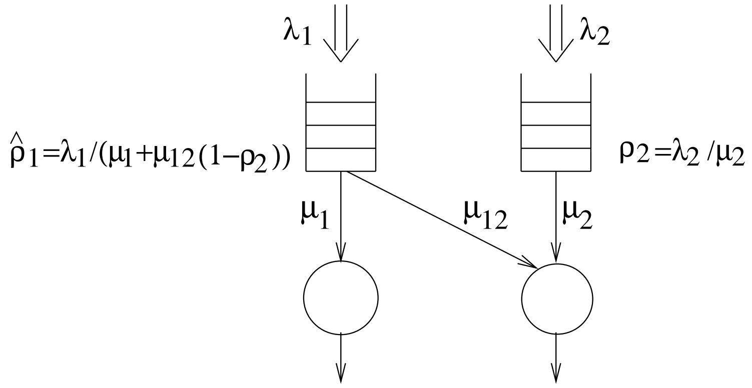


Figure 1: *A two server model.*

server 1. More recently, the *generalized $c\mu$ rule*, which is based on greedily minimizing the delay functions at any moment, has been applied to related models [18, 17]. However, in our model, the generalized $c\mu$ rule reduces to the $c\mu$ rule and hence has the same stability issues.

Squillante et. al. [23] have proposed a threshold-based policy that, under the right choice of threshold value, improves upon the $c\mu$ rule and guarantees stability whenever $\hat{\rho}_1 < 1$ and $\rho_2 < 1$. We refer to this threshold-based policy as the T1 policy, since it places a threshold value, T_1, on queue 1, so that server 2 only processes jobs from queue 1 (type 1 jobs) when there are at least T_1 jobs of type 1, or if queue 2 is empty. The rest of the time server 2 works on jobs from queue 2 (type 2 jobs). The motivation behind placing the threshold on queue 1 is that it 'reserves" a certain amount of work for server 1, preventing server 1 from being under-utilized and server 2 from being overloaded. More formally,

Definition 1 *The T1 policy is characterized by the following set of rules, all of which are enforced preemptively (preemptive-resume):*

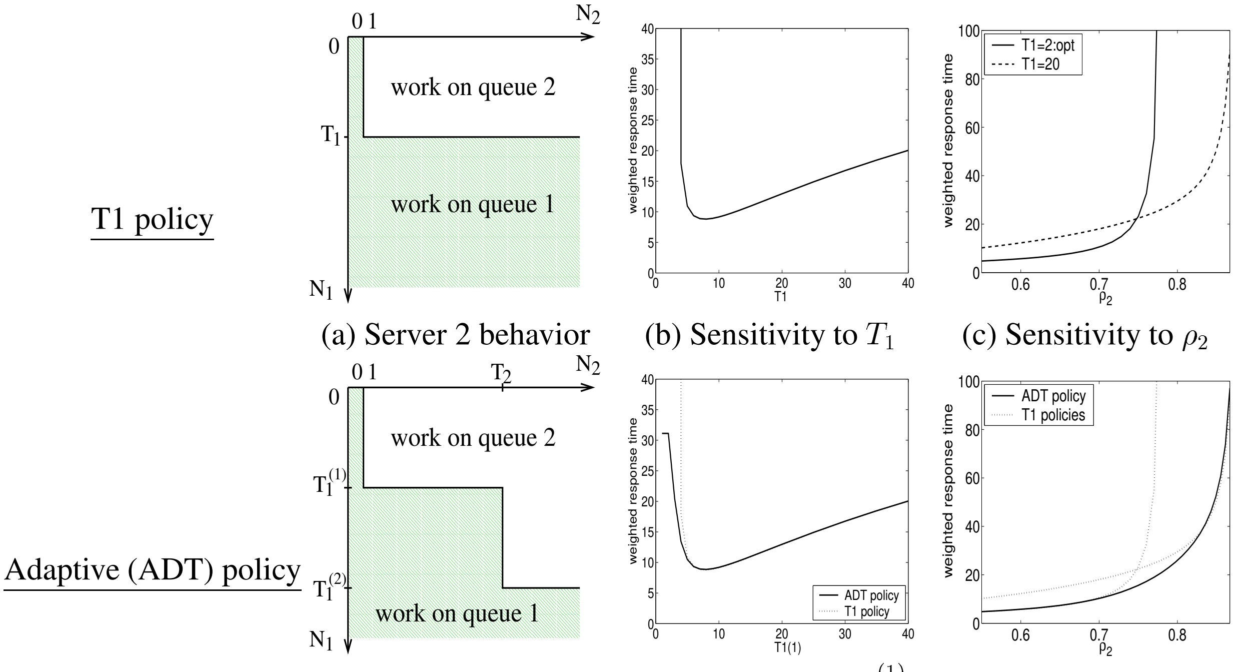
- *Server 1 serves only its own jobs.*
- *Server 2 serves jobs from queue 1 if either*

  1. $N_1 \geq T_1$, *or*
  2. $N_2 = 0$ & $N_1 \geq 2$

  Otherwise server 2 serves jobs from queue 2.[3]

Figure 2(a) shows which jobs server 2 processes as a function of the number of jobs in queue 1 (N_1) and queue 2 (N_2). Bell and Williams prove the optimality of the T1 policy for a model closely related to ours in the heavy traffic limit, where $\hat{\rho}_1$ and ρ_2 are close to 1 from below [2]. Williams conjectures that the T1 policy is optimal for more general models in the heavy traffic limit [29].

[3]To achieve maximal efficiency, we assume the following exceptions. When queue 2 is empty and queue 1 has only one job, i.e. $N_1 = 1$ and $N_2 = 0$, the job is assigned to the server with a higher service rate; namely, the job is processed by server 2 if and only if $\mu_1 < \mu_{12}$. Also, when $T_1 = 1$ and $N_1 = 1$, the job in queue 1 is processed by server 2 if and only if $\mu_1 < \mu_{12}$ regardless of the number of type 2 jobs.



(a) Server 2 behavior (b) Sensitivity to T_1 (c) Sensitivity to ρ_2

(d) Server 2 behavior (e) Sensitivity to $T_1^{(1)}$ (f) Sensitivity to ρ_2

Figure 2: *Comparison of the T1 policy and ADT policy.*

Despite its conjectured heavy traffic optimality, the T1 policy still has two problems. The first is how to determine T_1. Figure 2(b) shows the weighted average mean response time (weighted response time) as a function of T_1. The figure shows a common "V shape," where the optimal T_1 is very close to the T_1 values that lead to instability (infinite response time). This is quite problematic, as an exact analysis has never been derived. Squillante et. al. [23] only provide a coarse approximation; they then advocate choosing T_1 conservatively (higher than what their analysis predicts) at the cost of a higher mean response time. A second problem is the sensitivity of performance to a change in load. It is often the case that the load of a system changes over time, and even if it does not, estimating the correct average system load is often a difficult task [8]. Figure 2(c) shows the weighted response time as a function of ρ_2 (only λ_2 is changed) under T1 policies with two different threshold values. The T1 policy with optimal T_1 (=2) at an estimated load ($\rho_1 = 1.12$ and $\rho_2 = 0.6$) quickly becomes unstable at higher ρ_2 values (solid line). The T1 policy is not robust against a change in ρ_1, either. One can choose a higher T_1 (=20) to guarantee stability at higher loads, but this will result in worse performance at the estimated load (dashed line). Thus, the T1 policy exhibits a tradeoff between good performance at the estimated load and stability at higher loads.

In this paper we will introduce a new policy that allows us to get the best of both worlds, good performance at the estimated load *and* increased robustness. To derive our new policy, it helps to first better understand the T1 policy. Since the T1 policy has only been evaluated by simulation or by coarse approximation in the prior literature, we begin by introducing a computationally efficient and accurate evaluation method for threshold-based policies, including both the T1 policy and our new policy. Together these constitute contributions of this paper.

Contribution 1: We provide a computationally efficient and accurate analysis of the mean response time under the T1 policy as a function of job size and interarrival time distributions, server speeds, affinity, and weights. (See Sections 2-3.) Our analysis is technically an approximation, but can be made as accurate as desired, and appears exact when validated against simulation. Accurate analysis allows us to quickly find optimal threshold values for the T1 policy.

The same analysis technique can be applied to other threshold-based policies. For example, one might argue that the stability issue of the T1 policy with too small T_1 is resolved simply by placing an additional threshold, T_2, on queue 2, so that if the number of type 2 jobs, N_2, exceeds T_2, server 2 works on type 2 jobs regardless of the number of type 1 jobs. We refer to this policy as the T1T2 policy, since it operates as the T1 policy only when $N_2 \leq T_2$. This natural extension to the T1 policy surprisingly turns out to be in general *no better* than the T1 policy. That is, the optimal value of T_2 is usually ∞, reducing the T1T2 policy to the T1 policy. The intuition obtained through the extensive analysis of these policies leads us to propose a new, adaptive, threshold-based policy.

Contribution 2: We propose the Adaptive Dual-Threshold (ADT) policy that achieves performance similar to the optimal T1 policy ***and*** **is robust to changes in load** (See Section 4.) The key idea in the design of the ADT policy is the use of two thresholds, $T_1^{(1)}$ and $T_1^{(2)}$, on queue 1 with a threshold, T_2, on queue 2. The ADT policy behaves like the T1 policy with threshold $T_1^{(1)}$ if the number of jobs in queue 2 is less than T_2 and otherwise like the T1 policy with a higher threshold, $T_1^{(2)}$. Thus, in contrast to the T1T2 policy above, the ADT is *always* operating as a T1 policy, but unlike the standard T1 policy, the value of T_1 adapts, depending on the queue at server 2; the different thresholds on queue 1 allow server 2 to help queue 1 less when there are more type 2 jobs, preventing server 2 from becoming overloaded. This leads to the increased stability region. The dual thresholds also make the ADT policy adaptive to a change in load (changes in ρ_1 and ρ_2), in that it operates like the T1 policy with threshold $T_1^{(1)}$ at the estimated load and like the T1 policy with a higher threshold $T_1^{(2)}$ at a higher load.

Formally, the ADT policy is characterized by the following rule.

Definition 2 *If $N_2 \leq T_2$, the ADT policy operates as the T1 policy with threshold $T_1 = T_1^{(1)}$; otherwise, it operates as the T1 policy with threshold $T_1 = T_1^{(2)}$.*

Figure 2(d) shows which jobs server 2 processes under the ADT policy as a function of N_1 and N_2. When $T_1^{(1)} = T_1^{(2)}$ or when $T_2 = \infty$, the ADT policy reduces to the T1 policy with threshold $T_1^{(1)}$. Also, when $T_1^{(2)} = \infty$, the ADT policy reduces to the T1T2 policy.

The remaining figures illustrate the robustness of the ADT policy. Figure 2(f) shows the weighted response time as a function of ρ_2. Here, the ADT policy (solid line) performs just as well as the optimal T1 policy at the estimated load ($\rho_1 = 1.12$ and $\rho_2 = 0.6$), and has stability at higher ρ_2. We will show later that the ADT policy is also robust against changes in ρ_1. Figure 2(e) shows the weighted response time as a function of $T_1^{(1)}$. The figure suggests that once $T_1^{(2)}$ is chosen to guarantee the stability at the estimated load, the response time is finite for any $T_1^{(1)}$. As we will see, since the performance at the estimated load is relatively insensitive to $T_1^{(2)}$, we can choose a high $T_1^{(2)}$ to guarantee a large stability region. In addition, since the stability region is insensitive to $T_1^{(1)}$ and T_2, we can choose these values so that the performance at the estimated load is optimized.

In conclusion, the ADT policy is more robust than the T1 policy in two ways. (i) The response time under the ADT policy is less sensitive to changes in ρ_1 and ρ_2. (ii) The settings of the three threshold values under the ADT policy are less likely to lead to instability or increased response time, as compared to the T1 policy.

2 Analysis and validation of analysis

In this section, we first describe our analysis of the T1 policy. The analysis can be extended to other threshold-based policies, including the ADT policy, as illustrated in Section 2.2

2.1 Analysis of T1 policy

The difficulty in analyzing the mean response time under the T1 policy comes from the fact that the state space required to capture the system behavior grows infinitely in two dimensions; i.e., we need to track both the number of type 1 jobs and the number of type 2 jobs. In the literature, there are two types of approaches to analyze a process such as ours with two dimensionally infinite state space. The first approach is to resort to coarse approximation. For example, Squillante et. al. derive an approximate analysis of the T1 policy based on vacation models [23]. This type of analysis is computationally very efficient, but the accuracy of the solution is typically poor. For example, the error in the approximation by Squillante et. al. is unbounded, since it mispredicts the stability region. The second approach includes computational methods that can, in theory, be made as accurate as desired, but require more computational time. Such methods include reduction to a boundary value problem [6] and the matrix analytic method [16] possibly with state space truncation (e.g. [27]). Reduction to a boundary value problem is a mathematically elegant approach but often experiences numerical instability. The matrix analytic method can be computationally very expensive without state space truncation, and it is very difficult to determine where to truncate the state space to guarantee sufficient accuracy [3].

Our analysis of the T1 and ADT policies is a near exact method, which can be made as accurate as desired, based on the approach of dimensionality reduction that we introduce in [12]. Advantages of dimensionality reduction include its computational efficiency, accuracy, and simplicity; these allow us to extensively investigate the performance characteristics of the policies under consideration. The dimensionality reduction technique reduces a 2D-infinite Markov chain (hard to analyze) to a 1D-infinite Markov chain (easy to analyze) by using busy period transitions. Figure 3 shows the resulting 1D-infinite Markov chain for the T1 policy. This chain tracks the exact number of type 2 jobs. With respect to the number of type 1 jobs, the chain tracks this information only up to the point where there are $T_1 - 1$ jobs of type 1. At this point a type 1 arrival starts a "busy period[4]," which ends when there are once again $T_1 - 1$ jobs of type 1. During this busy period, both servers are working on type 1 jobs, and type 2 jobs receive *no* service. State (T_1^+, j) denotes there are *at least* T_1 jobs of type 1 and there are j type 2 jobs for $j = 0, 1, 2, \ldots$. *The key point is that there is no need to track the number of type 1 jobs during this busy period.*

[4]This busy period is equivalent to an M/M/1 busy period serving at rate $\mu_1 + \mu_{12}$.

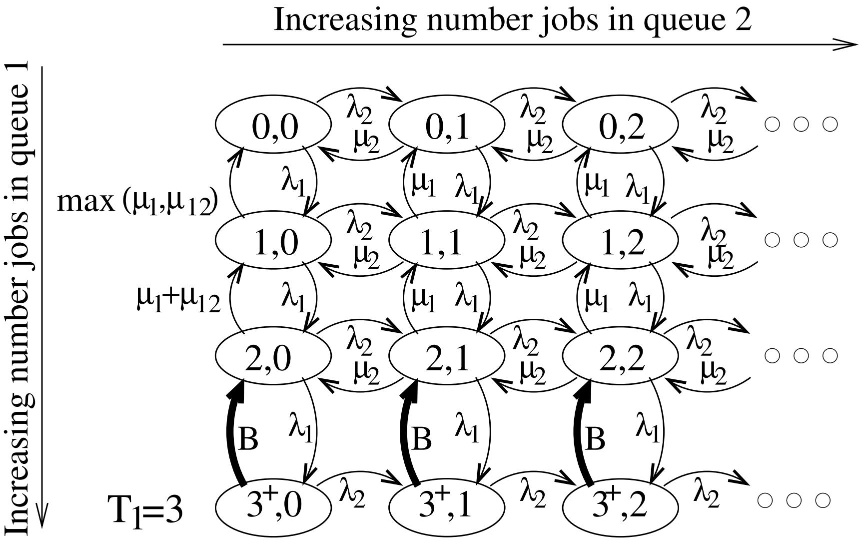


Figure 3: *Markov chain used for analyzing the T1 policy ($T_1 = 3$).*

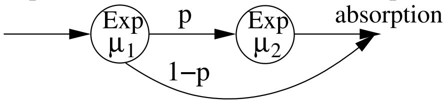
We approximate the distribution of the duration of this busy period by a 2-phase phase type (PH) distribution with Coxian representation[5] by matching the first three moments of the distributions [19]. We find that this suffices to achieve a high level of accuracy. Figure 4 validates our analysis against simulation; we show two of many cases we computed. Note that *generating a single data point via simulation requires 30 minutes*, since ten iterations, each with 1,000,000 events, are needed to stabilize. By contrast, *our analysis takes less than a second.*

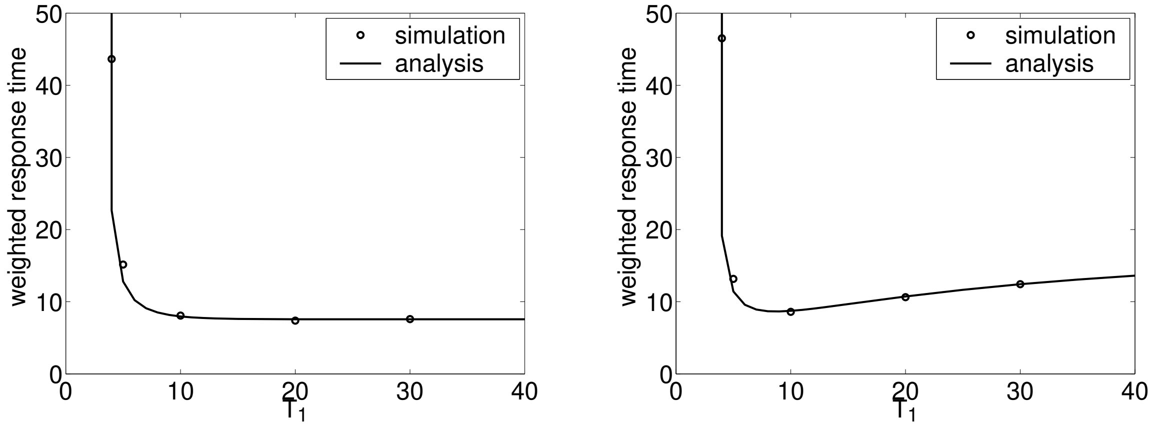
The limiting probabilities of the Markov chain in Figure 3 can be used to calculate the mean number of jobs of each type, $E[N_1]$ and $E[N_2]$, which in turn gives mean response time via Little's law [15]. The limiting probabilities of the 1D-infinite Markov chain can be obtained efficiently via the matrix analytic method [16]. Deriving $E[N_2]$ from the limiting probabilities is straightforward, since we track the exact number of type 2 jobs. We derive $E[N_1]$ by conditioning on the state of the chain. Let $E[N_1]_{ij}$ denote the expected number of type 1 jobs given that the chain is in state (i, j). For $i = 0, ..., T_1 - 1$, $E[N_1]_{ij} = i$ for all j. For $i = T_1^+$, $E[N_1]_{T_1^+,j}$ is the mean number of jobs in an M/M/1 system given that the service rate is the sum of the two servers, $\mu_1 + \mu_{12}$, and given that the system is busy, plus an additional T_1 jobs.

2.2 Analysis of ADT policy

The analysis of the mean response time under the ADT policy follows an approach similar to the analysis of the T1 policy. Figure 5 shows the 1D-infinite Markov chain used for analyzing the ADT policy. Again, the chain tracks the exact number of type 2 jobs, but with respect to the

[5] A PH distribution is the distribution of the absorption time in a continuous time Markov chain. The figure illustrates a 2-phase PH distribution with Coxian representation, where the ith state has exponentially-distributed sojourn time with rate μ_i for $i = 1, 2$. The absorption time is the sum of the times spent in each of the states, starting at state 1.





(a) $\mu_2 = \mu_1$ (b) $\mu_2 = \mu_1/4$

Figure 4: *Validation of analysis, where we set $\hat{\rho}_1 = 0.9$, $\rho_2 = 0.6$, and $\mu_{12} = \mu_1 = 1$.*

number of type 1 jobs it tracks this information only up to the point where there are $T_1^{(2)} - 1$ jobs. A type 1 arrival at this point starts a "busy period," which ends when there are $T_1^{(2)} - 1$ jobs of type 1. State $(T_1^{(2)^+}, j)$ denotes that there are *at least* $T_1^{(2)}$ jobs of type 1 and there are j jobs of type 2 for $j = 0, 1, 2, ...$ The mean response time is again obtained via the matrix analytic method.

3 Results: T1 Policy

Our analysis of Section 2 allows, for the first time, efficient and accurate analysis of the T1 policy. In this section we extensively evaluate the weighted average mean response time (weighted response time) under the T1 policy for various cases, and find the following characteristics of the T1 policy performance.

1. Setting the threshold T_1 higher yields a larger stability region.
2. When $c_1\mu_{12} \leq c_2\mu_2$, the optimal threshold with respect to minimizing the weighted response time is $T_1 = \infty$, which at the same time achieves the largest stability region. This is the policy of following the $c\mu$ rule, as in this case server 2 "prefers" to run its own jobs in a $c\mu$ sense.
3. When $c_1\mu_{12} > c_2\mu_2$, the optimal T_1 threshold is typically finite; in this case there is a tradeoff between good performance at the estimated load (ρ_1, ρ_2) and possible instability at higher ρ_1 and/or ρ_2. This is the case where server 2 "prefers" to run type 1 jobs in a $c\mu$ sense, but following the $c\mu$ rule leads to instability.
4. The lower the value of $c_1\mu_1$, the lower the optimal threshold tends to be, i.e. the closer it tends to be to the instability region. Hence, the tradeoff between good performance and stability is more dramatic at smaller $c_1\mu_1$.

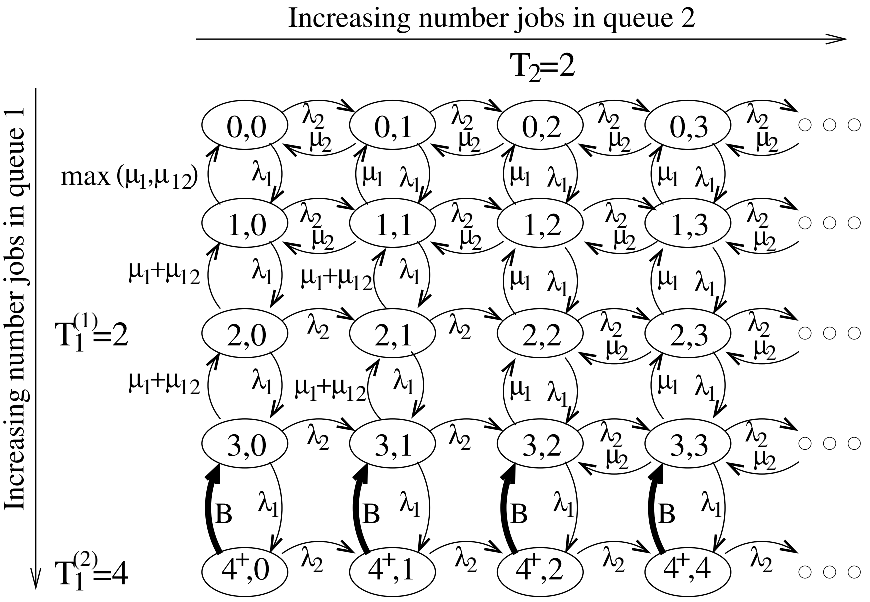


Figure 5: *Markov chain used for analyzing the ADT policy ($T_1^{(1)} = 2$, $T_1^{(2)} = 4$, $T_2 = 2$).*

3.1 Characterizing the optimal threshold

Figure 6 shows the weighted response time when type 1 jobs and type 2 jobs have the same weight, i.e., $c_1 = c_2 = 1$. Different rows correspond to different μ_1's and different columns correspond to different μ_2's, as labeled. Here, $\mu_{\mathbf{12}}$ **is fixed at 1**. The weighted response time is evaluated at three loads, $\hat{\rho}_1 = 0.8, 0.9, 0.95$ (only λ_1 is changed), and ρ_2 is fixed at 0.6 throughout.[6] We also discuss the effect of lower/higher ρ_2, though not shown in the figure. We consider relatively high loads, since system performance needs to be improved most in these cases.

Rule 1: If $\mathbf{c_1 = c_2}$ and $\mu_{\mathbf{12}} \leq \mu_{\mathbf{2}}$, $\mathbf{T_1 = \infty}$ is optimal. In the third and fourth columns of Figure 6, where $c_2\mu_2 \geq c_1\mu_{12} = 1$, the weighted response time is a nonincreasing function of T_1; hence $T_1 = \infty$ minimizes the weighted response time. As we will see in Section 3.2, $T_1 = \infty$ also maximizes the stability region. Hence, $T_1 = \infty$ (i.e., following the $c\mu$ rule) is the best choice with respect to both performance at the estimated load, (ρ_1, ρ_2), *and* stability at higher ρ_1 and/or ρ_2. The condition $\mu_{12} \leq \mu_2$ is achieved when type 1 jobs are large, type 2 jobs are small, and/or when type 1 jobs do not have good affinity with server 2 (e.g., when type 1 jobs require to access data stored locally at server 1). The following theorem formally characterizes Rule 1. We provide its proof in [20].

Theorem 1 *If $c_1 = c_2$ and $\mu_{12} \leq \mu_2$, $T_1 = \infty$ minimizes the overall mean response time.*

Rule 2: If $\mathbf{c_1 = c_2}$ and $\mu_{\mathbf{12}} > \mu_{\mathbf{2}}$, finite $\mathbf{T_1}$ is typically optimal. In the first and second columns of Figure 6, where $c_1\mu_{12} > c_2\mu_2$, the weighted response time has minimum at some finite T_1. Since a larger value of T_1 leads to a larger stability region, there is a tradeoff between

[6]Note that $\hat{\rho}_1 = 0.8, 0.9, 0.95$ corresponds to $\rho_1 = 5.92, 6.66, 7.03$ when $\mu_1 = 1/16$ (row 1), $\rho_1 = 2.08, 2.34, 2.47$ when $\mu_1 = 1/4$ (row 2), $\rho_1 = 1.12, 1.26, 1.33$ when $\mu_1 = 1$ (row 3), and $\rho_1 = 0.88, 0.99, 1.045$ when $\mu_1 = 4$ (row 4).

good performance at the estimated load, (ρ_1, ρ_2), and stability at higher ρ_1 and/or ρ_2. (Note that the curves have sharper "V shapes" in general at higher $\hat{\rho}_1$.) Choosing the right T_1 is further made difficult by the steep curve to the left of the optimal T_1: as T_1 becomes smaller, the weighted response time quickly diverges to infinity. Therefore, even when ρ_1 and ρ_2 are fixed and known, past works advised choosing T_1 conservatively, as they could not perform accurate analysis. Our accurate and efficient analysis now allows us to choose the optimal T_1 in such a situation. Though not shown, we have also investigated other values of ρ_2. When ρ_2 is lower (and hence ρ_1 is higher for a fixed $\hat{\rho}_1$), the optimal T_1 tends to become smaller, and hence the tradeoff between the performance at the estimated load and stability at higher loads is more significant. This makes intuitive sense, since at lower ρ_2, server 2 can help more. The condition $\mu_{12} > \mu_2$ is achieved when type 1 jobs are small, type 2 jobs are large, and/or in the pathological case when type 1 jobs have good affinity with server 2.

Rule 3: If $\mathbf{c_1 = c_2}$, lower $\boldsymbol{\mu_1}$ typically implies lower optimal threshold. In the upper rows in Figure 6, the optimal T_1's are smaller and at the same time the "V shapes" are sharper. Since smaller T_1 results in a smaller stability region and larger T_1 values significantly deteriorate the performance at the estimated load, the tradeoff between the performance and stability is most significant here. Small μ_1 is achieved when type 1 jobs are large or when server 1 is slow.

In the above rules we have assumed equal weights; Figure 7 shows the effect of changing the weights. Here c_2 is changed as labeled, c_1 is fixed at 1, and the service rates are now fixed at $\mu_1 = \mu_{12} = \mu_2 = 1$. In terms of $c\mu$ values, the figure corresponds to the third row of Figure 6. Observe that the curves in Figure 7 have shapes similar, but not identical, to the corresponding curves in Figure 6. The $c\mu$ rule provides a good basis but does miss some, potentially crucial, system changes. We find that as a whole, the above three rules continue to characterize the optimal T_1 values in the case of different weights when we replace μ_1 by $c_1\mu_1$, μ_{12} by $c_1\mu_{12}$, and μ_2 by $c_2\mu_2$ in the rules. In particular, when c_2 is small, i.e., type 1 jobs are more important, the optimal T_1 is smaller and the "V shape" is sharper, and hence the tradeoff between the performance at the estimated load, (ρ_1, ρ_2), and stability at higher ρ_1 and/or ρ_2 is most significant.

3.2 Stability of T1 policy

Figure 8 shows stability regions under the T1 policy as ρ_1 and ρ_2 vary; the queues are stable if and only if ρ_2 is below the curve. The figure illustrates that higher T_1 values yield a larger stability region, and in the limit of $T_1 = \infty$, the queues under the T1 policy are stable as long as $\hat{\rho}_1 < 1$ and $\rho_2 < 1$. The next theorem formally characterizes the stability of the T1 policy. We provide its proof in [20].

Evaluation of T1 policy

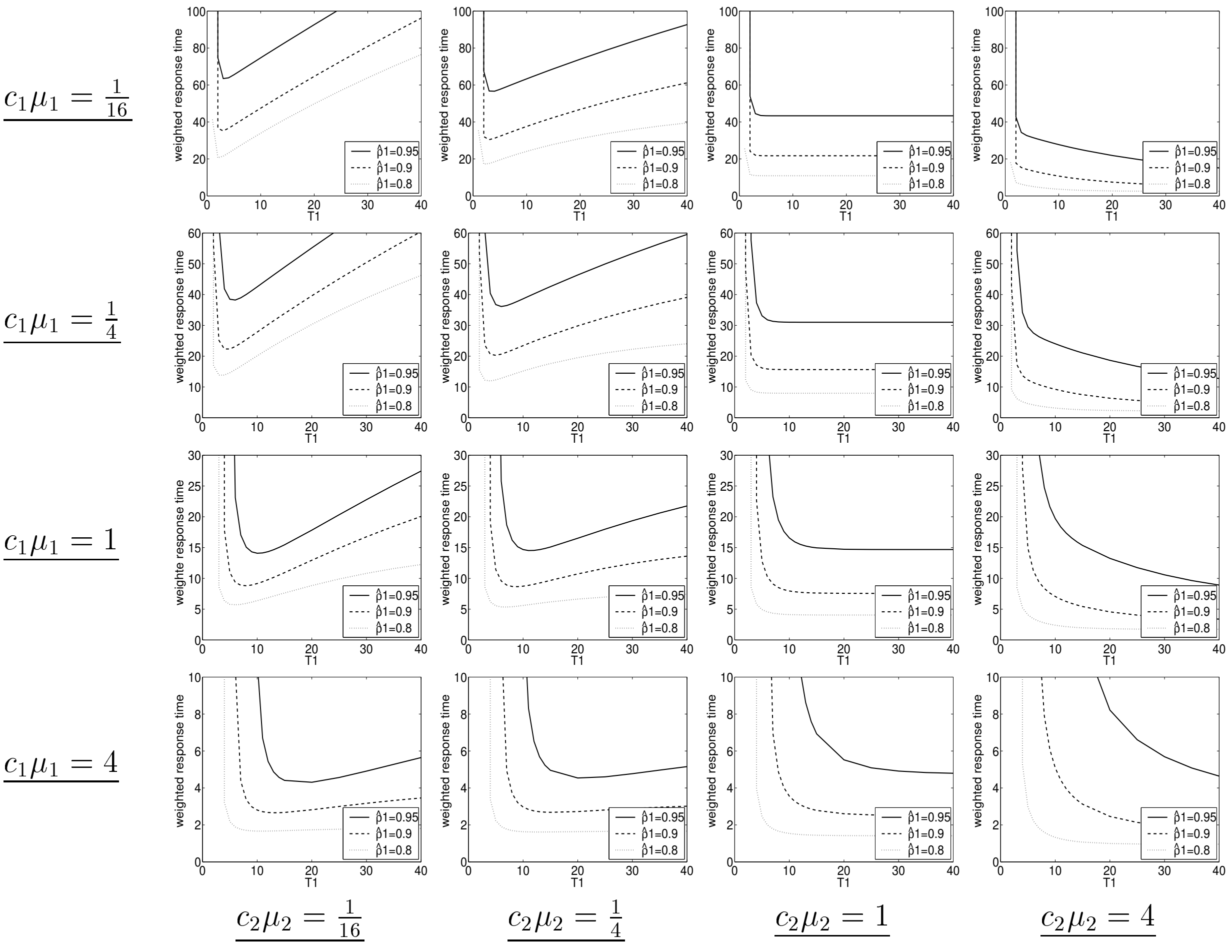


Figure 6: *The weighted response time under the T1 policy as a function of T_1, where type 1 jobs and type 2 jobs have the* **same weight**, $c_1 = c_2 = 1$. *Here,* $c_1\mu_{12} = 1$ *and* $\rho_2 = 0.6$ *are fixed. In all figures the three lines correspond to three different* $\hat{\rho}_1$*'s.*

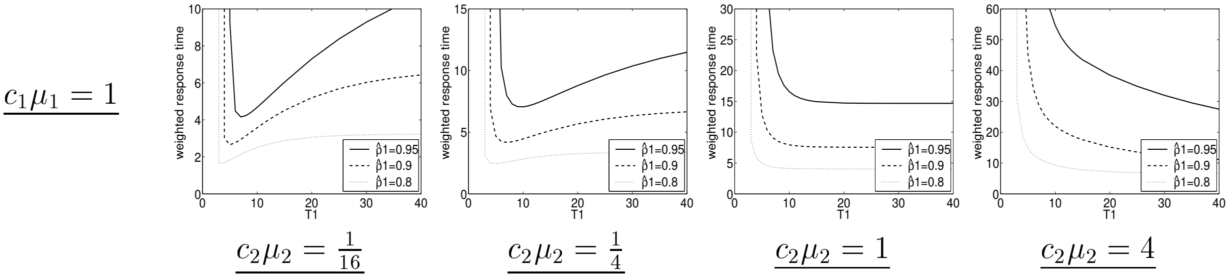
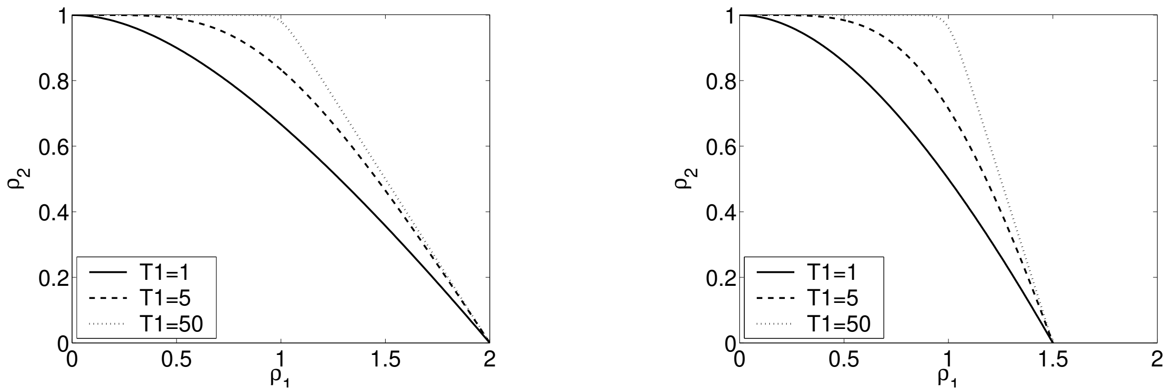


Figure 7: *The weighted response time under the T1 policy as a function of T_1, where type 1 jobs and type 2 jobs have* **different weights** *(service rates are fixed at* $\mu_1 = \mu_{12} = \mu_2 = 1$*). Here,* $c_1\mu_{12} = 1$ *and* $\rho_2 = 0.6$ *are fixed. The three lines correspond to three different* $\hat{\rho}_1$*'s.*



(a) $\mu_{12} = \mu_1$ (b) $\mu_{12} = \frac{\mu_1}{2}$

Figure 8: *Stability region of the T1 policy for different threshold values. Upper bounds on ρ_2 are plotted as functions of ρ_1.*

Theorem 2 **Queue 1** *is stable under the T1 policy if and only if $\lambda_1 < \mu_1 + \mu_{12}$.* **Queue 2** *is stable under the T1 policy, when $T_1 > 1$, if and only if*

$$\rho_2 \quad < \quad \begin{cases} \frac{1-\rho_1^{T_1}}{1-\rho_1^{T_1}+\frac{(1-\rho_1)\rho_1^{T_1-1}}{\frac{1}{\rho_1}+\frac{\mu_{12}}{\lambda_1}-1}} & \text{if } \rho_1 \neq 1, \\ \\ \frac{1+\rho_1(T_1-1)}{1+\rho_1(T_1-1)+\frac{\rho_1^{T_1-1}}{\frac{1}{\rho_1}+\frac{\mu_{12}}{\lambda_1}-1}} & \text{if } \rho_1 = 1. \end{cases} \tag{1}$$

When $T_1 = 1$ and $\mu_1 \geq \mu_{12}$, queue 2 is stable if and only if equation (1) holds with $T_1 = 2$. When $T_1 = 1$ and $\mu_1 < \mu_{12}$, queue 2 is stable if and only if

$$\rho_2 < \frac{1}{1 + \frac{\lambda_1}{\mu_{12}} + \frac{\lambda_1/\mu_{12}}{1/\rho_1 + \mu_{12}/\lambda_1 - 1}}.$$

Corollary 1 *The right hand side of equation (1) is an increasing function of T_1; i.e., stability increases with T_1.*

4 Results: ADT policy

In this section we examine the adaptive dual-threshold (ADT) policy, which achieves both good performance at the estimated load and stability at higher loads. Recall from Section 3 that small T_1 achieves good performance at the expense of stability, and large T_1 achieves stability at the expense of good performance. The ADT policy places two thresholds, $T_1^{(1)}$ and $T_1^{(2)}$, on queue 1, and as shown in this section:

1. Performance at the estimated load is well characterized by $T_1^{(1)}$, and
2. Stability is characterized by $T_1^{(2)}$.

Thus we get the best of both worlds. Since the ADT policy requires specifying three thresholds, $T_1^{(1)}$, $T_1^{(2)}$, and T_2, one might want to avoid searching the space of all possible triples for the optimal settings. We show that threshold values set by the following sequential heuristic can achieve performance comparable to the optimal settings.

1. Set $T_1^{(1)}$ as the optimal T_1 of the T1 policy at the estimated load.
2. Choose $T_1^{(2)}$ so that it achieves stability in a desired range of load.
3. Find T_2 such that the policy provides both good performance and stability.

4.1 Stability of the ADT policy

We first consider the stability of the ADT policy. At high enough ρ_1 and ρ_2, N_2 usually exceeds T_2 and the policy behaves similar to the T1 policy with $T_1 = T_1^{(2)}$; the stability condition for the ADT policy is in fact the same as the stability condition for the T1 policy when T_1 is replaced by $T_1^{(2)}$. The following corollary can be proved in a similar way as Theorem 2.

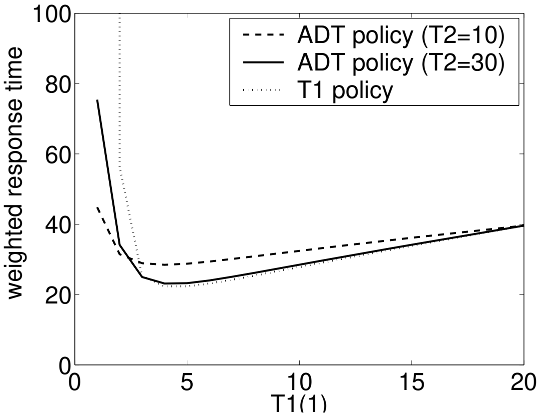
Corollary 2 *The stability condition (necessary and sufficient) for the ADT policy is given by the stability condition for the T1 policy (Theorem 2) by replacing T_1 by $T_1^{(2)}$.*

4.2 Characterizing the optimal thresholds

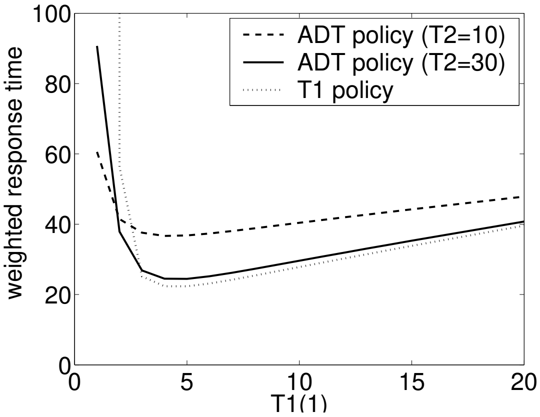
Corollary 2 suggests that $T_1^{(2)}$ should be chosen so that the policy can achieve stability in a desired range of load. One might argue that setting $T_1^{(2)}$ too high degrades the performance at the estimated load, as does the T1 policy with large T_1 threshold; however, it turns out that this is *not* the case for the ADT policy. Figure 9 shows the weighted average mean response time (weighted response time) as a function of $T_1^{(1)}$ (solid or dashed lines); it also shows the weighted response time under the T1 policy as a function of T_1 (dotted lines). We set $T_1^{(2)}$ and T_2 as labeled. The figure shows that choosing high $T_1^{(2)}$ (=40) degrades the performance at the estimated load ($\rho_1 = 2.34$, $\hat{\rho}_1 = 0.9$, and $\rho_2 = 0.6$) only if T_2 is too small (dashed line). With a sufficiently large T_2, the ADT policy achieves performance very comparable to the optimal T1 policy at the estimated load (solid line). Note also that for $T_1^{(1)}$ values below the optimal T_1, the performance of the ADT remains stable, in contrast to the simple T1 policy.

Figure 9 also suggests that when T_2 is appropriately set, the weighted response time is minimized by setting $T_1^{(1)}$ close to the T_1 that minimizes the weighted response time, achieving performance very close to the optimal T1 policy at the estimated load.

However, determining the appropriate T_2 is a nontrivial task. If T_2 is set too low, the ADT policy behaves like the T1 policy with threshold $T_1 = T_1^{(2)}$, degrading the performance at the estimated load, since $T_1^{(2)}$ is larger than the optimal T_1. If T_2 is set too high, the ADT policy behaves like the T1 policy with threshold $T_1 = T_1^{(1)}$. This worsens the performance at loads



$\underline{T_1^{(2)} = 20}$



$\underline{T_1^{(2)} = 40}$

Figure 9: *The weighted response time under the ADT policy as a function of $T_1^{(1)}$ (solid or dashed lines) and the weighted response time under the T1 policy as a function of T_1 (dotted lines). Here we assume $\hat{\rho}_1 = 0.9$, $\rho_2 = 0.6$, $c_1\mu_1 = 1/4$, $c_1\mu_{12} = 1$, and $c_2\mu_2 = 1/16$, corresponding to the second graph in the first column of Figure 6.*

higher than the estimated load. Although a larger stability region is *guaranteed* by setting $T_1^{(2)}$ higher than the optimal T_1, the weighted response time at higher loads can be quite high, albeit finite.

Our efficient and accurate analysis of the ADT policy in Section 2 can be used as the kernel of a search algorithm to determine a good value of T_2 swiftly and easily, once we determined $T_1^{(1)}$ and $T_2^{(2)}$. Figure 10 shows the weighted response time for different $c\mu$ values, as a function of $\hat{\rho}_1$ in column (a) and as a function of ρ_2 in column (b). Dashed lines show the weighted response time under the T1 policy, using the threshold T_1 that minimizes the weighted response time at the estimated load ($\hat{\rho}_1 = 0.9$ and $\rho_2 = 0.6$). Dotted lines show the weighted response time under the T1 policy with $T_1 = 20$. Solid lines show the weighted response time under the ADT policy, where $T_1^{(1)}$ is set at the optimal T_1 at the estimated load, $T_1^{(2)} = 20$, and T_2 is chosen so that it achieves good performance both at the estimated load and at higher $\hat{\rho}_1$ and higher ρ_2. We find "good" T_2 values manually by trying a few different values, which takes us a few minutes.

Figure 10 shows that the ADT policy has at least as good performance as the better of the T1 policies throughout the range of loads ($\hat{\rho}_1$ and ρ_2). When μ_1 is large relative to μ_{12} (bottom rows), the difference in the weighted response time at the estimated load is small between the optimal T_1's and larger T_1's. Therefore a high threshold, such as $T_1 = 20$, is a reasonable choice in this case, implying that a good T_2 is typically small. This case corresponds to the case when type 1 jobs have good affinity at server 1, or the case when server 1 is faster.

When μ_1 is small relative to μ_{12} (top rows), stability region is less sensitive to T_1, since small T_1 can achieve relatively large stability region. Therefore, the optimal T_1 at the estimated load is a good choice, and T_2 is typically large. These rows correspond to the case when server 2 is faster, or type 1 jobs have good affinity on server 2.

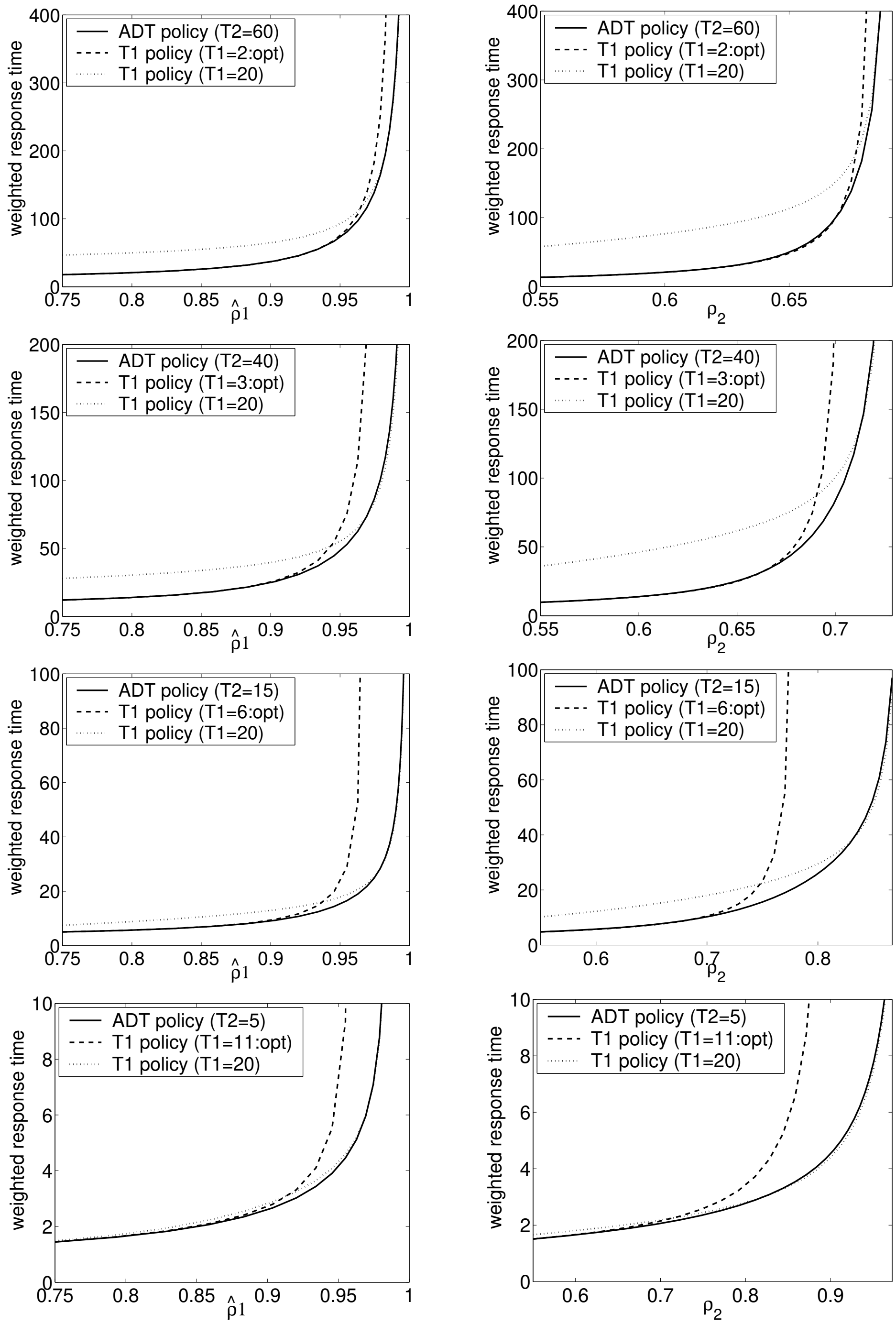
The advantage of the ADT policy over the T1 policy is most significant when μ_1 is close

$c_1\mu_1 = \frac{1}{16}$

$c_1\mu_1 = \frac{1}{4}$

$c_1\mu_1 = 1$

$c_1\mu_1 = 4$



(a) sensitivity to $\hat{\rho}_1$ (b) sensitivity to ρ_2

Figure 10: *The weighted response time under the ADT policy (solid lines) and T1 policy (dashed and dotted lines) as functions of (a) $\hat{\rho}_1$ (only λ_1 is changed; $\rho_2 = 0.6$) and (b) ρ_2 (only λ_2 is changed; $\hat{\rho}_1 = 0.8$). Threshold T_1 is chosen so that the weighted response time is minimized at $\hat{\rho}_1 = 0.8$ and $\rho_2 = 0.6$ (dashed lines) or $T_1 = 20$ (dotted lines). For the ADT policy, $T_1^{(1)}$ is set as the optimal T_1, $T_1^{(2)} = 20$, and T_2 is chosen so that it achieves good performance and stability. All graphs assume $c_1\mu_{12} = 1$, $c_2\mu_2 = 1/16$, and $c_1 = c_2 = 1$, corresponding to the first column of Figure 6.*

to μ_{12}. In this case, small T_1 provides good performance at low $\hat{\rho}_1$ and at low ρ_2 but has a significantly smaller stability region, while large T_1 provides a larger stability region but has significantly worse performance at low $\hat{\rho}_1$ and at low ρ_2. The ADT policy performs just as well as the T1 policy with small T_1 at low $\hat{\rho}_1$ and ρ_2, just as well as the T1 policy with large T_1 at high $\hat{\rho}_1$ and ρ_2, and better than any of the two T1 policies at intermediate values of $\hat{\rho}_1$ and ρ_2.

5 Conclusion

Providing good performance at the estimated environment parameters such as loads, arrival rates, and job sizes has been a central goal in designing allocation policies and schedules in computer systems. However, estimating environment parameters is a difficult task, and furthermore these parameters typically fluctuate over time.

We propose the Adaptive Dual-Threshold (ADT) policy that provides good performance in the estimated environment and yet is also robust against changes in load. The key idea in the design of the ADT policy is the use of dual thresholds; autonomously choosing the right threshold depending on the system state, and hence adapting to changes in the environment. We show the effectiveness of the ADT policy in the case of two servers; however, the ADT policy could be applied to any more general distributed computing system where a threshold-based policy is effective. We hope that the intuition obtained in our study of the two server case is also useful in choosing the correct threshold values in more general systems.

We also provide a computationally efficient and accurate performance analysis framework that is widely applicable to threshold-based policies, including the ADT policy. This analysis allows us to determine threshold values so that the policy provides good performance *and* robustness. Our analysis is useful in investigating the characteristics of threshold-based policies in general; in fact we have studied many threshold-based policies, including the T1 and T1T2 policies. The intuition obtained in the study of these degenerate cases led us to propose the ADT policy.

We have described our analysis in the case of two servers and two queues with Poisson arrivals and exponential jobs sizes, but this can be extended to more general cases. Job size and interarrival time distributions can be extended to general distributions using PH distributions as approximations (see [11]). It is also straightforward to extend our analysis to more than two servers but with only two queues. Extension to more than two queues is not trivial, but in certain cases the recursive dimensionality reduction that we propose in [28] may be applied.

References

[1] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, and S. Krishnakumar. Oceano - SLA based management of a computing utility. In *Proceedings of the IFIP/IEEE Symposium on Integrated Network Management*, pages 855–868, May 2001.

[2] S. Bell and R. Williams. Dynamic scheduling of a system with two parallel servers in heavy traffic with complete resource pooling: Asymptotic optimality of a continuous review threshold policy. *Annals of Probability*, 11:608–649, 2001.

[3] L. Bright and P. Taylor. Calculating the equilibrium distribution in level dependent quasi-birth-and-death processes. *Stochastic Models*, 11:497–514, 1995.

[4] J. Chase, L. Grit, D. Irwin, J. Moore, and S. Sprenkle. Dynamic virtual clusters in a grid site manager. In *Proceedings of the International Symposium on High Performance Distributed Computing*, pages 90–103, June 2003.

[5] J. S. Chase, D. C. Anderson, P. N. Thankar, and A. M. Vahdat. Managing energy and server resources in hosting centers. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 103–116, October 2001.

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

[7] D. Cox and W. Smith. *Queues*. Kluwer Academic Publishers, 1971.

[8] P. A. Dinda and D. R. O'Hallaron. Host load prediction using linear models. *Cluster Computing*, 3(4):265–280, 2000.

[9] L. Green. A queueing system with general use and limited use servers. *Operations Research*, 33(1):168–182, 1985.

[10] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems*, 15(3):253–285, 1997.

[11] M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, and M. Squillante. Task assignment with cycle stealing under central queue. In *Proceedings of The 23rd International Conference on Distributed Computing Systems (ICDCS 2003)*, pages 628–637, May 2003.

[12] M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, and M. Squillante. Task assignment with cycle stealing under immediate dispatch. In *Proceedings of The Fifteenth ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 274–285, June 2003.

[13] J. Harrison. Heavy traffic analysis of a system with parallel servers: Asymptotic optimality of discrete review policies. *Annals of Applied Probability*, 8(3):822–848, 1998.

[14] J. Harrison and M. Lopez. Heavy traffic resource pooling in parallel server systems. *Queueing Systems*, 33(4):339–368, 1999.

[15] L. Kleinrock. *Queueing Systems, Volume I: Theory*. A Wiley-Interscience Publication, 1975.

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

[17] A. Mandelbaum and A. Stolyar. Scheduling flexible servers with convex delay costs: Heavy traffic optimality of the generalized $c\mu$-rule. *Operations Research*, to appear.

[18] J. V. Miegham. Dynamic scheduling with convex delay costs: the generalized $c\mu$ rule. *Annals of Applied Probability*, 5(3):809–833, 1995.

[19] T. Osogami and M. Harchol-Balter. A closed-form solution for mapping general distributions to minimal PH distributions. In *Proceedings of the Performance TOOLS*, pages 200–217, September 2003.

[20] T. Osogami, M. Harchol-Balter, A. Scheller-Wolf, and L. Zhang. An adaptive threshold-based policy for sharing servers with affinities. Technical Report CMU-CS-04-112, School of Computer Science, Carnegie Mellon University, 2004.

[21] R. Schumsky. Approximation and analysis of a call center with specialized and flexible servers. *working paper*, 2004.

[22] M. Squillante and D. Lazowska. Using processor-cache affinity information in shared-memory multiptocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):131–143, 1993.

[23] M. Squillante, C. Xia, D. Yao, and L. Zhang. Threshold-based priority policies for parallel-server systems with affinity scheduling. In *Proceedings of the IEEE American Control Conference*, pages 2992–2999, June 2001.

[24] M. S. Squillante and R. D. Nelson. Analysis of task migration in shared-memory multiprocessors. In *Proceedings of the ACM SIGMETRICS*, pages 143–155, May 1991.

[25] M. S. Squillante, C. H. Xia, and L. Zhang. Optimal scheduling in queuing network models of high-volume commercial web sites. *Performance Evaluation*, 47(4):223–242, 2002.

[26] D. Stanford and W. Grassman. The bilingual server system: A queueing model featuring fully and partially qualified servers. *INFOR*, 31(4):261–277, 1993.

[27] D. Stanford and W. Grassmann. Bilingual server call centers. In D. McDonald and S. Turner, editors, *Analysis of Communication Networks: Call Centers, Traffic and Performance*. American Mathematical Society, 2000.

[28] A. Wierman, T. Osogami, M. Harchol-Balter, and A. Scheller-Wolf. Analyzing the effect of prioritized background tasks in multiserver systems. Technical Report CMU-CS-03-213, School of Computer Science, Carnegie Mellon University, 2003.

[29] R. Williams. On dynamic scheduling of a parallel server system with complete resource pooling. In D. McDonald and S. Turner, editors, *Analysis of Communication Networks: Call Centers, Traffic and Performance*. American Mathematical Society, 2000.

A Proofs of theorems

A proof of Theorem 1

By Little's law, minimizing the mean number of jobs in system results in minimizing the mean response time. Thus, it is sufficient to prove that the number of jobs completed under the T1 policy with $T_1 = \infty$ is stochastically larger than those completed under the T1 policy with $T_1 < \infty$ at any moment. Let $N^{\rm inf}(t) = (N_1^{\rm inf}(t), N_1^{\rm inf}(t))$ be the joint process of the number of jobs in queue 1 and queue 2, respectively, at time t when $T_1 = \infty$; Let $N^{\rm fin}(t) = (N_1^{\rm fin}(t), N_1^{\rm fin}(t))$ be defined analogously for $T_1 < \infty$. With $T_1 = \infty$, server 2 processes type 2 jobs as long as there are type 2 jobs, and thus $N_1^{\rm inf}(t)$ is stochastically larger than $N_1^{\rm fin}(t)$ for all t. This implies that the number of jobs completed by sever 1 is stochastically smaller when $T_1 < \infty$ than when $T_1 = \infty$ at any moment, since server 1 is work-conserving.

As long as server 2 is busy, the number of jobs completed by sever 2 is stochastically smaller when $T_1 < \infty$ than when $T_1 = \infty$ at any moment, since $\mu_{12} \leq \mu_2$. The only time that server 2 becomes idle is when there are no jobs to be processed in either queue; at these epochs $N_1^{\rm inf}(t) = N_1^{\rm inf}(t) = N_1^{\rm inf}(t) = N_1^{\rm inf}(t) = 0$, and the number of jobs completed (either by server 1 or by server 2) becomes the same for $T_1 = \infty$ and $T_1 < \infty$. This implies that the number of jobs completed (either by server 1 or by server 2) under the T1 policy with $T_1 = \infty$ is stochastically larger than that completed under the T1 policy with $T_1 < \infty$.

A proof of Theorem 2

We prove only the case when $T_1 > 1$. The case when $T_1 = 1$ can be proved in a similar way. Let $N = (N_1, N_2)$ be the joint process of the number of jobs in queue 1 and queue 2, respectively. Consider a process $\tilde{N} = (\tilde{N}_1, \tilde{N}_2)$, where $\tilde{N}$ behaves the same as N except that it has no transition from $\tilde{N}_2 = 1$ to $\tilde{N}_2 = 0$. Then, N_1 and N_2 are stochastically smaller than $\tilde{N}_1$ and $\tilde{N}_2$, respectively. It suffices to prove the stability of $\tilde{N}$.

First consider $\tilde{N}_1$. The expected length of a "busy period," during which $\tilde{N}_1 \geq T_1$, is finite if and only if $\lambda_1 < \mu_1 + \mu_{12}$. This proves the stability condition for queue 1.

The strong law of large numbers can be used to show that the necessary and sufficient condition for stability of queue 2 is $\rho_2 < F$, where F denotes the time average fraction of time that server 2 processes jobs from queue 2. Below, we derive F. Consider a semi-Markov process of $\tilde{N}_1$, where the state space is (0,1,2,...,$T_1 - 1$,T_1^+). The state n denotes there are n jobs in queue 1 for $n = 0, 1, ..., T_1 - 1$, and the state T_1^+ denotes there are *at least* T_1 jobs in queue 1. The expected sojourn time is $1/\lambda_1$ for state 0, $1/(\lambda_1 + \mu_1)$ for states $n = 1, ..., T_1 - 1$, and $b = \frac{1/(\mu_1+\mu_{12})}{1-\lambda_1/(\mu_1+\mu_{12})}$ for state T_1^+, where b is the mean duration of the busy period in an M/M/1 queue, where the arrival rate is λ_1 and the service rate is $\mu_1 + \mu_{12}$. The limiting probabilities for the corresponding *embedded* Markov chain are $\pi_n = (1 + \rho_1)\rho_1^{n-1}\pi_0$ for $n = 1, ..., T_1 - 1$ and $\pi_{T_1^+} = \rho_1^{T_1}\pi_0$, where

$$\pi_0 = \begin{cases} \frac{1}{1+\rho_1^{(T_1-1)}+(1+\rho_1)\frac{1-\rho_1^{(T_1-1)}}{1-\rho_1}} & \text{if } \rho_1 \neq 1 \\ \frac{1}{1+\rho_1^{(T_1-1)}+(1+\rho_1)(T_1-1)} & \text{if } \rho_1 = 1. \end{cases}$$

In $\tilde{N}$, server 2 can work on queue 2 if and only if $\tilde{N}_1 < T_1$. So, the fraction of time that server 2 can work on queue 2 is

$$\begin{aligned} F &= \frac{\pi_0/\lambda_1 + (1-\pi_0-\pi_{T_1^+})/(\lambda_1+\mu_1)}{\pi_0/\lambda_1 + (1-\pi_0-\pi_{T_1^+})/(\lambda_1+\mu_1) + b\pi_{T_1^+}} \\ &= \begin{cases} \frac{1-\rho_1^{T_1}}{1-\rho_1^{T_1}+\frac{(1-\rho_1)\rho_1^{T_1-1}}{\frac{1}{\rho_1}+\frac{\mu 12}{\lambda_1}-1}} & \text{if } \rho_1 \neq 1 \\ \\ \frac{1+\rho_1(T_1-1)}{1+\rho_1(T_1-1)+\frac{\rho_1^{T_1-1}}{\frac{1}{\rho_1}+\frac{\mu 12}{\lambda_1}-1}} & \text{if } \rho_1 = 1. \end{cases} \end{aligned}$$

This proves the stability condition for queue 2.