Exploring Threshold-based Policies for Load Sharing*

Takayuki Osogami
Computer Science Department
Carnegie Mellon University
osogami@cs.cmu.edu

Mor Harchol-Balter
Computer Science Department
Carnegie Mellon University
harchol@cs.cmu.edu

Alan Scheller-Wolf
Tepper School of Business
Carnegie Mellon University
awolf@andrew.cmu.edu

Li Zhang
Thomas J. Watson Research Center
IBM Research
zhangli@us.ibm.com

Abstract

We consider the problem of how to design resource allocation policies that both provide good performance at predicted environmental conditions and are robust against changes or misprediction of the environmental conditions. We evaluate various common threshold-based allocation policies within a simple model, where there is a clear tradeoff between the (conflicting) goals of good performance and robustness. We then propose and evaluate a new threshold-based policy, ADT (adaptive dual thresholds), that achieves both the desired goals.

1 Introduction

A common problem in computer and communication systems is deciding how to allocate resources (e.g. CPU time and bandwidth) among jobs. A good (resource) allocation policy that maximizes system performance, e.g. with respect to mean response time and throughput, often has parameters that need to be tuned to achieve the best performance. Since the optimal settings of the parameters typically depend on environmental conditions such as system loads, an allocation policy whose parameters are chosen to achieve the best performance in a certain environment can provide poor performance when the environment changes or when the prediction of the environment was wrong.

The objective of this paper is to design and study characteristics of various allocation policies in a simple model, where there is a clear tradeoff between good performance and robustness against changes and misprediction in loads. The study in this simple model provides lessons that are useful in designing allocation policies in more complex systems.

Our model consists of two servers and two queues, as shown in Figure 1. Jobs arrive at queue 1 and queue 2 according to Poisson processes with rates λ_1 and λ_2, respectively. Jobs have exponentially distributed service demands; however, the running time of a job may also depend on the affinity between the particular server and the particular queue. Hence, we assume that server 1 processes jobs in queue 1 (type 1 jobs) with rate

*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.

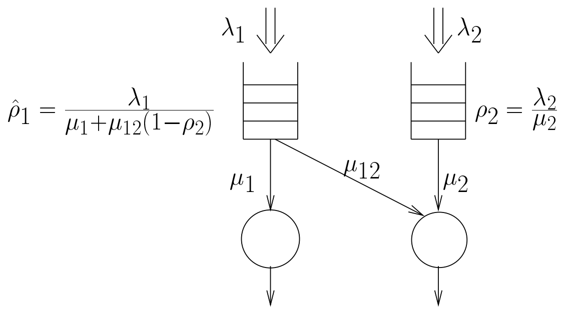


Figure 1: *A two server model.*

μ_1 (jobs/sec), while server 2 can process type 1 jobs with rate μ_{12} (jobs/sec) and can process jobs in queue 2 (type 2 jobs) with rate μ_2 (jobs/sec). We define $\rho_1 = \lambda_1/\mu_1$, $\rho_2 = \lambda_2/\mu_2$, and $\hat{\rho}_1 = \lambda_1/(\mu_1 + \mu_{12}(1 - \rho_2))$. Note that $\rho_2 < 1$ and $\hat{\rho}_1 < 1$ are necessary for the queues to be stable under any allocation policy, since the maximum rate at which type 1 jobs can be processed is μ_1, from server 1, plus $\mu_{12}(1 - \rho_2)$, from server 2.

In this paper, we design and evaluate allocation policies in the model in Figure 1 with respect to two objectives. First, we seek to minimize the weighted average mean response time, $c_1p_1E[R_1] + c_2p_2E[R_2]$, where c_i is the weight (importance) of type i jobs, $p_i = \lambda_i/(\lambda_1 + \lambda_2)$ is the fraction of type i jobs, and $E[R_i]$ is the mean response time[1] of type i jobs, for $i = 1, 2$. Second, we want our policy to be robust against misprediction and changes in loads, ρ_1 and ρ_2. In this paper, we focus on threshold-based allocation policies, since these are common and natural in our model. Note that the optimal allocation policy is not known in our model, despite the fact that it has been investigated in numerous papers [1, 3, 4, 5, 7, 8] (see also references in [6]).

We start, in Section 2, by considering two common allocation policies. The first policy (T1 policy) places a threshold, T_1, on queue 1, whereby server 2 serves type 1 jobs only when the length of queue 1 exceeds T_1 (or serve 2 is idle). The second policy (T2 policy) places a threshold, T_2, on queue 2, whereby server 2 serves type 1 jobs only when the length of queue 2 is *below* T_2. Only coarse approximations exist for analyzing response time under the T1 and T2 policies. Hence, we introduce a near-exact analysis technique in [6], which is also applicable to all the allocation policies that we investigate in this paper. Our analysis demonstrates a tradeoff between good performance (low response time) at predicted load and robustness across loads in the T1 and T2 policies. This tradeoff motivates us, in Section 3, to introduce two new allocation policies (the T1T2 policy and the ADT policy), both of which are based on the idea of using multiple thresholds. While these two new allocation policies appear similar in their definition, it turns out that their characteristics are very different. In particular, we show that the ADT policy is able to achieve both good performance at predicted load and robustness.

2 Evaluating simple threshold-based policies

2.1 T1 and T2 policies

The T1 policy is motivated by some shortcomings of the cμ rule [2]. Recall that the cμ rule biases in favor of jobs with high c (high importance) and high μ (small expected

[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.

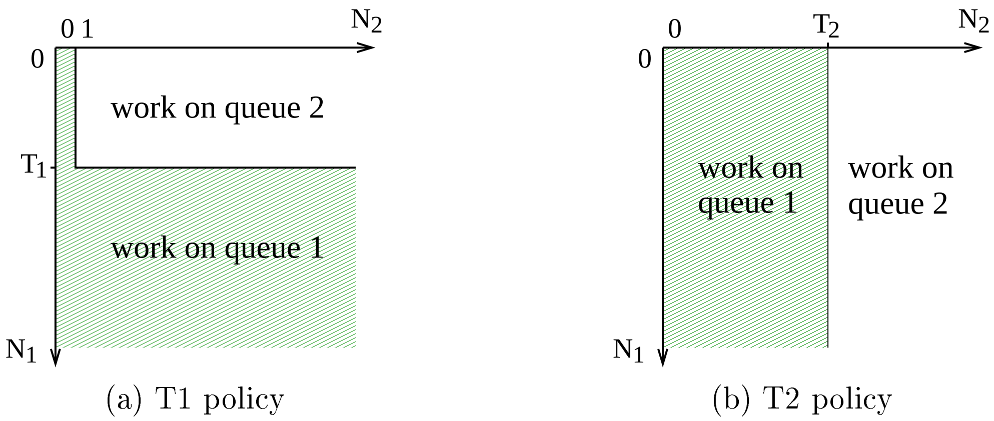


Figure 2: *Figures show whether server 2 works on jobs from queue 1 or queue 2 as a function of N_1 and N_2, under (a) the T1 policy and (b) the T2 policy.*

size). Applying the cμ rule to our setting translates to letting a server process jobs from the nonempty queue with the highest $c\mu$ value. Under the $c\mu$ rule, server 2 serves type 1 jobs (rather than type 2 jobs) if $c_1\mu_{12} > c_2\mu_2$, or queue 2 is empty. The $c\mu$ rule is provably optimal when server 1 does not exist [2]. However Squillante et. al. [7] as well as Harrison [4] 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 type 1 jobs even when many jobs are built up at queue 2, leading to instability in queue 2 and under-utilization of server 1.

Squillante et. al. [7] and Williams [8] have independently 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 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 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 *Let N_1 (respectively, N_2) denote the number of jobs at queue 1 (respectively, queue 2). The T1 policy with parameter T_1 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 (i) $N_1 \geq T_1$ or (ii) $N_2 = 0$ & $N_1 \geq 2$. Otherwise, server 2 serves jobs from queue 2.*[2]

Figure 2(a) shows the jobs processed by server 2 as a function of N_1 and N_2 under the T1 policy. Note that the T1 policy with $T_1 = 1$ is the same as the cμ rule when $c_1\mu_{12} > c_2\mu_2$, and the T1 policy with $T_1 = \infty$ is the same as the cμ rule when $c_1\mu_{12} < c_2\mu_2$. 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 [1]. In the T1 policy, the higher T_1 values yield the 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 $\rho_2 < 1$. More formally, we prove the following theorem in [6]:

[2]To achieve maximal efficiency, we assume the following exceptions. When $N_1 = 1$ and $N_2 = 0$, 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.

Theorem 1 *Under the T1 policy with parameter $T_1 < \infty$, queue 1 is stable if and only if $\lambda_1 < \mu_1 + \mu_{12}$, and queue 2 is stable if and only if*

$$\rho_2 < \frac{1-\rho_1^{T_1}}{1-\rho_1^{T_1} + \frac{(1-\rho_1)\rho_1^{T_1-1}}{\frac{1}{\rho_1}+\frac{\mu_{12}}{\lambda_1}-1}},$$

when $T_1 > 1$ and $\rho_1 \neq 1$. (See [6] for the case of $T_1 = 1$ or $\rho_1 = 1$.)

An alternative threshold-based policy that guarantees stability whenever $\hat{\rho}_1 < 1$ and $\rho_2 < 1$ is the T2 policy. The T2 policy places a threshold value, T_2, on queue 2, such that server 2 processes type 1 jobs only when there are less than T_2 jobs of type 2, thus preventing server 2 from being overloaded. More formally,

Definition 2 *The T2 policy with parameter T_2 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 $N_2 < T_2$. Otherwise server 2 serves jobs from queue 2.*[3]

Figure 2(b) shows the jobs processed by server 2 as a function of N_1 and N_2 under the T2 policy. Recall that the T1 policy guarantees stability whenever $\hat{\rho}_1 < 1$ and $\rho_2 < 1$ *provided* that T_1 is chosen appropriately. By contrast, the T2 policy guarantees stability whenever $\hat{\rho}_1 < 1$ and $\rho_2 < 1$ for *any* finite T_2. More formally, the following theorem holds, which we state without proof:

Theorem 2 *Under the T2 policy with $T_2 < \infty$, queue 1 is stable if and only if $\hat{\rho}_1 < 1$, and queue 2 is stable if and only if $\rho_2 < 1$.*

2.2 Comparison of T1 and T2 policies

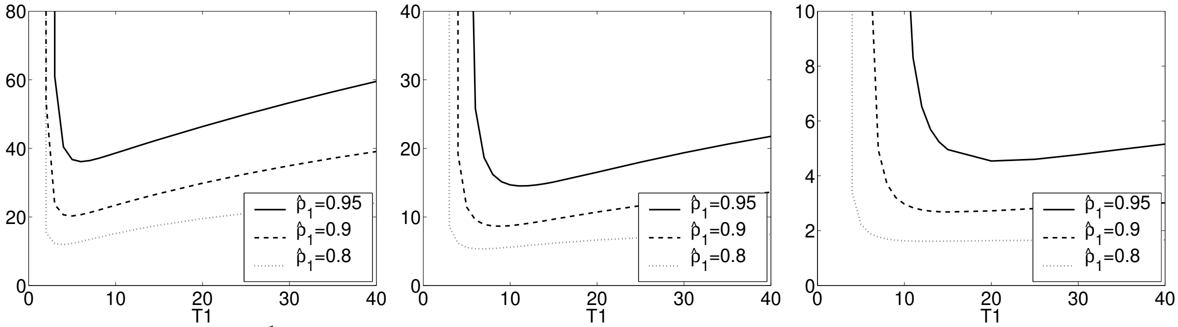
In this section, we study characteristics of the T1 policy and the T2 policy by evaluating the weighted mean response time under various settings. In [6], we introduce a computationally efficient and near-exact analysis of the mean response time under the T1 and T2 policies, and this analysis enables us to study the T1 and T2 policies extensively. In this paper, we limit our focus on the case where type 1 jobs and type 2 jobs have the same weight, i.e. $c_1 = c_2 = 1$; for a general case of $c_1 \neq c_2$, see [6].

When $c_1 = c_2$ and $\mu_{12} \leq \mu_2$, we prove in [6] that $T_1 = \infty$ is the optimal choice for the T1 policy and $T_2 = 1$ is the optimal choice for the T2 policy with respect to both performance at the estimated load and robustness. Thus, the T1 and T2 policies with the optimal threshold values become the same under this setting (i.e. they both follow the $c\mu$-rule: server 2 works on jobs from queue 1 only when queue 2 is empty). Hence, below, we limit our attention to the case of $\mu_{12} > \mu_2$. Note that condition $\mu_{12} > \mu_2$ is achieved when type 1 jobs are small and type 2 jobs are large (in the general case of $c_1 \neq c_2$, condition $c_1\mu_{12} > c_2\mu_2$ is also achieved when type 1 jobs are more important than type 2 jobs) and/or in the pathological case when type 1 jobs have good affinity with server 2.

Figure 3 shows the weighted mean response time (overall mean response time) under the T1 policy (top row) and the T2 policy (bottom row). Different columns correspond to different μ_1's. Here, $c_1\mu_{12} = 1$ and $c_2\mu_2 = \frac{1}{4}$ are fixed. The overall mean response time

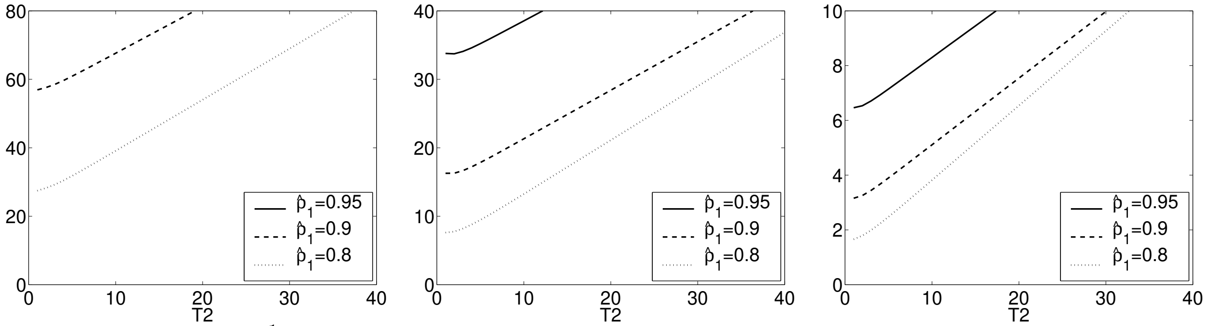
[3] When $N_1 = 1$ and $N_2 = 0$, we allow the same exception as in the T1 policy.

T1 policy



(a) $c_1\mu_1 = \frac{1}{4}$ (b) $c_1\mu_1 = 1$ (c) $c_1\mu_1 = 4$

T2 policy



(d) $c_1\mu_1 = \frac{1}{4}$ (e) $c_1\mu_1 = 1$ (f) $c_1\mu_1 = 4$

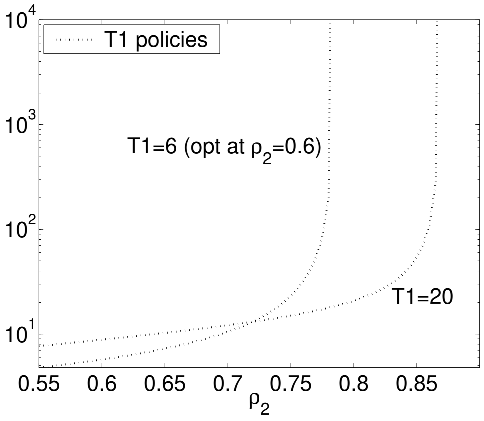
Figure 3: *The overall mean response time under the T1 and T2 policies as a function of T_1 and T_2. Here, $c_1 = c_2 = 1$, $c_1\mu_{12} = 1$, $c_2\mu_2 = \frac{1}{4}$, and $\rho_2 = 0.6$ are fixed. When $c_1\mu_1 = \frac{1}{4}$ (in the left column) and $\hat{\rho}_1 = 0.95$, the overall mean response time under the T2 policy is over 100 for all T_2, and does not appear in the figure.*

is evaluated at three loads, $\hat{\rho}_1 = 0.8, 0.9, 0.95$ (only λ_1 is changed)[4], and ρ_2 is fixed at 0.6 throughout. See [6] for discussion on the other values of ρ_2.

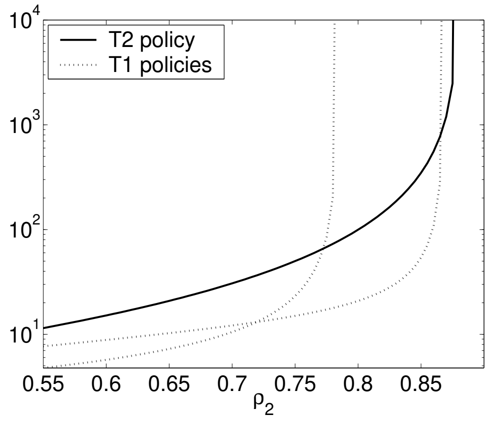
The top row of Figure 3 shows that the overall mean response time under the T1 policy is minimized at some finite T_1, and that the optimal T_1 depends on environmental conditions such as load ($\hat{\rho}_1$) and job sizes (μ_1). By Theorem 1, a larger value of T_1 leads to a larger stability region, and hence there is a tradeoff between good performance at the estimated load, $(\hat{\rho}_1, \rho_2)$, which is achieved at smaller T_1, and stability at higher $\hat{\rho}_1$ and/or ρ_2, which is achieved at larger T_1. Note also that the curves have sharper "V shapes" in general at higher $\hat{\rho}_1$, which make it difficult to choose the right T_1, since the overall mean response time quickly diverges to infinity, as T_1 becomes smaller.

The bottom row of Figure 3 shows that the overall mean response time under the T2 policy is minimized at $T_2 = 1$ or small T_2. Since choosing either $T_2 = 1$ or small T_2 minimizes the overall mean response time at the estimated load and still provides the maximum stability region, there is no tradeoff. However, observe that the overall mean response time under the T2 policy with the optimal T_2 can be much higher than that under the T1 policy with the optimal T_1.

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



(a) T1 policy



(b) T2 policy

Figure 4: *Overall mean response time under the T1 policy and the T2 policy ($T_2 = 1$) as a function of ρ_2, where $c_1 = c_2 = 1$, $c_1\mu_1 = c_1\mu_{12} = 1$, $c_2\mu_2 = \frac{1}{16}$, and $\rho_1 = 1.12$ are fixed.*

Figure 4(a) highlights the tradeoff between the performance at the estimated load and the robustness against changes and misprediction in load in the T1 policy, plotting the overall mean response time as a function of ρ_2 (only λ_2 is changed). When $\rho_2 = 0.6$, $T_1 = 6$ is the optimal choice, and overall mean response time is lower with $T_1 = 6$ than with $T_1 = 20$. If it turns out that $\rho_2 = 0.8$ is the actual load, then the T1 policy with $T_1 = 6$ leads to instability (infinite overall mean response time), while the T1 policy with $T_1 = 20$ still gives finite and low overall mean response time. In the above sense, the T1 policy is not robust against misprediction or changes in load. 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. Thus, the T1 policy exhibits a tradeoff between good performance at the estimated load and robustness against changes and misprediction of load.

Since the T2 policy is typically optimal with $T_2 = 1$ and the maximum stability region is guaranteed with $T_2 = 1$, one might expect that the T2 policy has robustness against misprediction or changes in load. Figure 4(b) shows the overall mean response time under the T2 policy with $T_2 = 1$ as a function of ρ_2. Although the T2 policy is more robust than the T1 policy in the sense that it can guarantee finite overall mean response time for a wider range of load, the figure suggests that the finite overall mean response time can be very high under the T2 policy.

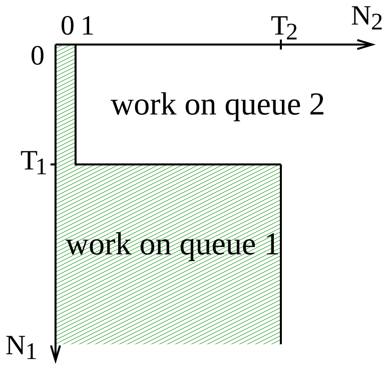
3 Designing new robust threshold-based policies

3.1 T1T2 policy

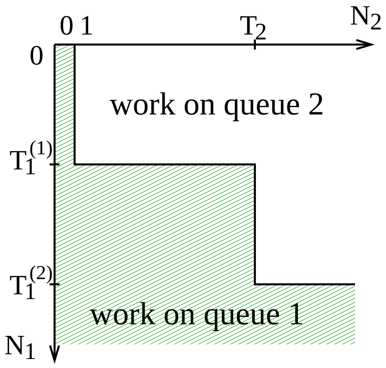
One might argue that the stability issue of the T1 policy with small optimal T_1 is resolved simply by placing an additional threshold, T_2, on queue 2, so that if the length of queue 2, N_2, exceeds T_2, server 2 works on type 2 jobs regardless of the length of queue 1, thus preventing queue 2 from becoming unstable. We refer to this policy as the T1T2 policy, since it operates as the T1 policy only when $N_2 \le T_2$. More formally,

Definition 3 *The T1T2 policy with parameters T_1 and T_2 is characterized by the following set of rules, all of which are enforced preemptively (preemptive-resume):*

- *Server 1 serves only its own jobs.*



(a) T1T2 policy



(b) ADT policy

Figure 5: *Figures show whether server 2 works on jobs from queue 1 or queue 2 as a function of N_1 and N_2 under (a) the T1T2 policy and (b) the ADT policy.*

- *Server 2 serves jobs from queue 1 if either (i) $N_1 \geq T_1$ & $N_2 < T_2$ or (ii) $N_2 = 0$ & $N_1 \geq 2$. Otherwise, server 2 serves jobs from queue 2.*[5]

Figure 5(a) shows the jobs processed by server 2 a function of N_1 and N_2 under the T1T2 policy. The stability region of the T1T2 policy is the same as that of the T2 policy. We state the following theorem without proof:

Theorem 3 *Under the T1T2 policy with parameters T_1 and $T_2 < \infty$, queue 1 is stable if and only if $\hat{\rho}_1 < 1$, and queue 2 is stable if and only if $\rho_2 < 1$.*

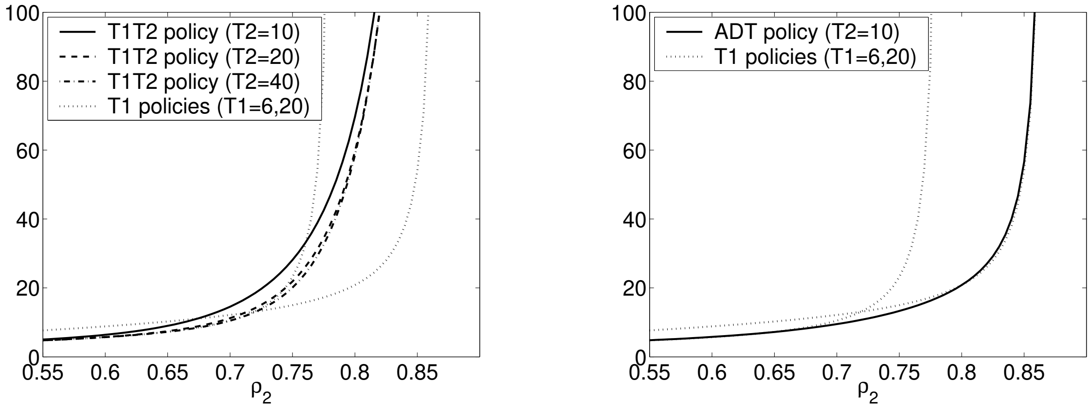
Figure 6(a) shows the overall mean response time under the T1T2 policy as a function of ρ_2, where $T_1 = 6$ and $T_2 = 10$, 20, or 40. Recall that the T1 policy achieves its lowest overall mean response time given $\rho_2 = 0.6$ when $T_1 = 6$. The T1T2 policy with $T_1 = 6$ is designed to provide a wider stability region with the near-optimal overall mean response time at $\rho_2 = 0.6$. In fact, when $\rho_2 = 0.6$, the overall mean response time under the T1T2 policy with $T_1 = 6$ is comparable to that under the T1 policy with $T_1 = 6$ for a range of T_2. For higher ρ_2 (specifically, $\rho_2 > 0.76$), the T1T2 policy (with $T_1 = 6$) provides lower overall mean response time than the T1 policy with $T_1 = 6$, hence being more robust. However, the range of load for which the T1T2 policy improves upon the T1 policy is limited. For example, when $\rho_2 = 0.8$, the overall mean response time under the T1T2 policy with any value of T_2 is significantly higher than the T1 policy with $T_1 = 20$.

The inadequacy of the T1T2 policy is primarily due to the fact that the T1T2 policy operates like the T2 policy at higher load, but the performance of the T2 policy is typically poor at any load as compared to the optimal T1 policy. This motivates us to introduce a new policy, the ADT policy, which *always* operates as a T1 policy.

3.2 ADT policy

The key idea in the design of the adaptive dual threshold (ADT) policy is the use of two thresholds, $T_1^{(1)}$ and $T_1^{(2)}$, both on queue 1 together with a threshold, T_2, on queue 2. The ADT policy behaves like the T1 policy with threshold $T_1^{(1)}$ if the length of 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, the ADT policy is *always* operating as a T1 policy, but unlike the standard T1 policy, the value of T_1 adapts, depending on the length of queue 2.

[5]When $N_1 = 1$ and $N_2 = 0$, we allow the same exception as in the T1 policy.



(a) T1T2 policy (b) ADT policy

Figure 6: *Overall mean response time under (a) the T1T2 policy and (b) the ADT policy as a function of* ρ_2*, where* $c_1 = c_2 = 1$*,* $c_1\mu_1 = c_1\mu_{12} = 1$*,* $c_2\mu_2 = \frac{1}{16}$*, and* $\rho_1 = 1.12$*.*

We will see that the ADT policy is more robust than the T1 policy against changes and misprediction in loads due to the dual thresholds on queue 1. First, the dual 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. Second, the dual thresholds make the ADT policy adaptive to changes in load ($\hat{\rho}_1$ 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 4 *The ADT policy with parameters* $T_1^{(1)}$*,* $T_1^{(2)}$*, and* T_2 *operates as the T1 policy with parameter* $T_1 = T_1^{(1)}$ *if* $N_2 \le T_2$*; otherwise, it operates as the T1 policy with parameter* $T_1 = T_1^{(2)}$*.*

Figure 5(b) shows the jobs processed by server 2 under the ADT policy as a function of N_1 and N_2. At high enough $\hat{\rho}_1$ and ρ_2, N_2 usually exceeds T_2, and the policy behaves similarly to the T1 policy with $T_1 = T_1^{(2)}$. Thus, the stability condition for the ADT policy is the same as that for the T1 policy when T_1 is replaced by $T_1^{(2)}$. In [6], we prove the following theorem:

Theorem 4 *The stability condition (necessary and sufficient) for the ADT policy with parameters* $T_1^{(1)}$*,* $T_1^{(2)}$*, and* T_2 *is given by the stability condition for the T1 policy with parameter* $T_1 = T_1^{(2)}$ *(Theorem 1).*

Figure 6(b) illustrates the robustness of the ADT policy, showing the overall mean response time under the ADT policy as a function of ρ_2. It is observed that (i) performance at the estimated load ($\rho_2 = 0.6$ in Figure 6(c)) is well characterized by $T_1^{(1)}$, and (ii) stability is characterized by $T_1^{(2)}$ (recall Theorem 4). Also, the ADT policy achieves at least as good performance as the *better* of the T1 policies with two different T_1 values throughout the range of ρ_2. We show in [6] that the ADT policy is also robust against changes in $\hat{\rho}_1$.

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. In choosing the thresholds of the ADT policy in Figure 6, we have followed the following sequential heuristic:

1. Set $T_1^{(1)}$ as the optimal T_1 value for 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.

We find that the performance at the estimated load is relatively insensitive to $T_1^{(2)}$, and hence we can choose a high $T_1^{(2)}$ to guarantee a large stability region (see [6] for details). Also, 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. 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 in the T1 policy. 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 higher than the estimated load. Although a larger stability region is *guaranteed* by setting $T_1^{(2)}$ higher than the optimal T_1 in the T1 policy, the overall mean response time at higher loads can be quite high, albeit finite. In plotting Figure 6, we find "good" T_2 values manually by trying a few different values, which takes only a few minutes.

4 Conclusion

In this paper, we design and evaluate various threshold-based resource allocation policies in a simple model of two servers and two queues. This provides us with lessons that are useful in designing resource allocation policies in more complex systems. The study of simple allocation policies, the T1 policy and the T2 policy, reveals the tradeoff between good performance at the estimated environmental conditions versus robustness against changes and misprediction of the environmental conditions. For example, we have seen that when the threshold value is chosen appropriately, the performance of the T1 policy is no worse than or very close to the best performance achieved by all the other allocation policies studied in this paper. However, the optimal threshold value for the T1 policy depends on the environmental conditions, and a threshold value that works for the current load may cause instability under higher loads. On the other hand, the T2 policy guarantees the maximum stability region and has more robustness, but its performance is usually poor. The superiority in performance of the T1 policy over the T2 policy brings up another interesting point: it is better to determine when help is provided based on the "beneficiary" queue length rather than the "donor" queue length.

An obvious "fix" for the lack of robustness in the T1 policy is to use an additional threshold to guarantee stability at higher load. This is the idea behind the design of the T1T2 policy. It turns out, however, that the improvement in robustness in the T1T2 policy is marginal. This is primarily due to the fact that the T1T2 policy operates like the T2 policy at higher load, and the performance of the T2 policy is typically poor at any load. That is, letting the both queues have control (the T1T2 policy) is not much better than letting the beneficiary queue alone have control (the T1 policy). The inadequacy of the T1T2 policy motivates us to propose a new allocation policy, the ADT policy.

Unlike the T1T2 policy, the ADT policy always operates as a T1 policy, adapting its threshold value to changes in environmental conditions. A difficulty in designing such an adaptive allocation policy is detection of changes in the environmental conditions or precise prediction of the environmental conditions. In our model, we are able to "detect" the changes or misprediction in the environmental conditions by observing the length of queue 2, N_2. In particular, the ADT policy uses a threshold value that is appropriate at low load when N_2 is low, and it uses a threshold value that is appropriate at high load when N_2 is high. It turns out that the performance of the ADT policy is better than or very close to the two T1 policies with different threshold values; that is, the ADT policy can provide good performance at estimated environmental conditions and is also robust. We conjecture that a policy that uses more thresholds on queue 1 and chooses an appropriate threshold depending on N_2 would provide better performance across a wider range of load, at the expense of additional complexity.

Finally, we provide some guidelines for designing resource allocation policies with good performance and robustness for more complex computer and communication systems. A first step would be to design an allocation policy that can provide good performance at the estimated environmental conditions (the T1 policy in our model). It may help to consider a simpler approximate model. A second step would be to find some indicator, within the system, of the changes/misprediction of the environmental conditions (N_2 in our model). If there is no such internal indicator, we would need to design one. A last step would be to find an appropriate mapping from the internal state (e.g., N_2) to the parameter (e.g., T_1) of the good allocation policy (e.g., the T1 policy). Towards this end, it would be helpful to have analysis technique (e.g., our analysis technique in [6] used throughout this paper) that allows us to evaluate the allocation policy swiftly and accurately at various environmental conditions.

References

[1] 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 Applied Probability*, 11:608–649, 2001.

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

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

[4] J. M. 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.

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

[6] 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.

[7] 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.

[8] 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.