

Threshold-Based Priority Policies for Parallel-Server Systems with Affinity Scheduling

Mark S. Squillante[†], Cathy H. Xia[†], David D. Yao^{‡,1}, Li Zhang[†]

[†] IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598.

[‡] IEOR Department, Columbia University, New York, NY 10027.

1 Introduction

Across a wide spectrum of parallel and distributed computer systems, it can be the case that a job is processed more efficiently on one server than on another [6]. Due to the inevitability of imbalances in the amount of work assigned to each server in these computing environments, there exists a dynamic scheduling tradeoff between keeping the workload balanced and scheduling jobs where they are processed most efficiently [7]. A fundamental understanding of this scheduling tradeoff is of great theoretical interest, and it also has important applications in the design of control policies for high-performance computer and telecommunication systems, as well as manufacturing systems.

We study the dynamic scheduling of such multiclass parallel-server stochastic networks, where the service rate depends upon both the job class being processed and the server being used to execute a job from this class. Our objective is to identify the control policy that dynamically allocates the parallel servers among waiting jobs so as to minimize the long-run average inventory cost, i.e., $\min \sum_k c_k L_k$ where c_k is the inventory cost per unit time for class k jobs and L_k is the long-run average number of class k jobs in the system. While it can be shown that the $c\mu$ -rule which matches jobs to the best server that can process them (in terms of lowest cost) is optimal in the context of the corresponding fluid network [8], studies have also established that such a greedy, $c\mu$ -rule type of policy does not always work well in the original stochastic setting. Quite to the contrary, it may even result in an unstable system.

Here, we propose a general class of threshold-based priority policies that works as follows. When a server becomes available and has to select a queue to process next, it first checks all queues to which thresholds have been assigned and for which the number of jobs is above the corresponding threshold value. The server then processes a job from the queue with the highest $c\mu$ value. If all of these queues are below their corresponding thresholds, the server then checks all the nonempty queues and selects a job from the queue with the highest $c\mu$ value. We develop an algorithm to determine the set of queues where thresholds should be placed under this class of priority policies.

To obtain the corresponding optimal threshold values, we develop closed-form approximations for some key performance measures, based on a combination of priority-queue and server-vacation models. Extensive numerical comparisons between this type of approximations and simulation have resulted in very good agreement. Moreover, the results have demonstrated that this class of threshold-based priority policies performs very well when proper threshold values are chosen.

The remainder of this paper is organized as follows. In §2 we study the dynamic scheduling of multiclass parallel-server stochastic networks and propose a general class of threshold-based priority policies. A number of examples are considered in §3 to illustrate the use of our algorithm for determining the placement of thresholds and to quantify the benefits of the proposed policies. Finally, in §4 we derive approximate formulas for the queue lengths under our threshold-based priority policies and we illustrate how these formulas can be used to obtain optimal threshold values.

2 General Threshold Policies

2.1 Motivation: Two-Queue Case

Consider the system in Figure 1 with two servers and two queues under Poisson arrivals and exponential service times (and all the usual independence assumptions). Let \mathcal{S}_i , \mathcal{Q}_j and \mathcal{C}_k denote server i , queue j and class k , respectively. The model parameters are: $c_1 = 100$, $c_2 = 1$, $\lambda_1 = 1.3\rho$, $\lambda_2 = 0.4\rho$, $\mu_{11} = \mu_{22} = 1$, $\mu_{21} = 0.5$, $\mu_{12} = 0$; where c_k is the unit inventory cost for \mathcal{C}_k jobs; λ_j is the \mathcal{C}_j arrival rate, with parameter $\rho > 0$; μ_{11} and μ_{22} are the “affinity” service rates, meaning jobs in \mathcal{Q}_1 (resp. \mathcal{Q}_2) are best served by \mathcal{S}_1 (resp. \mathcal{S}_2); whereas μ_{21} (resp. μ_{12}) is the service rate for \mathcal{S}_2 (resp. \mathcal{S}_1) to process jobs in \mathcal{Q}_1 (resp. \mathcal{Q}_2). In particular, $\mu_{12} = 0$ indicates that \mathcal{S}_1 is *not allowed* to process any job in \mathcal{Q}_2 .

This system has been considered in [1] with deterministic service times; refer also to [9]. It is known that when $\rho = 0.95$, for instance, following the $c\mu$ -rule will result in an unstable system. This is also the case when service times are exponential [8]. We note, however, that there exist policies that can maintain a stable system in this case; e.g., if \mathcal{S}_2

¹Supported in part by an NSF grant ECS-9705392.

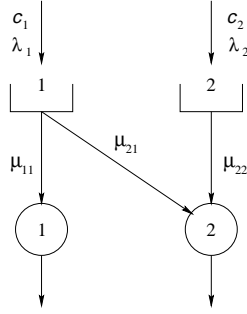


Figure 1: A Two-Server, Two-Queue Example

gives higher priority to Q_2 , the system is stable.

Observe that in this example, S_2 is shared by both Q_1 and Q_2 , while Q_1 is shared by both S_1 and S_2 . This gives rise to an “N” shape structure where the diagonal link connects the shared server with the shared queue. The main reason for the $c\mu$ -rule being unstable is that the shared S_2 devotes too much effort to help Q_1 , thus leaving insufficient capacity to process Q_2 while also causing S_1 capacity to be underutilized by C_1 jobs.

To avoid such behavior, or specifically, to prevent S_2 from being too greedy in its execution of C_1 jobs, we propose a policy that sets a threshold T at Q_1 for S_2 , such that S_2 will serve jobs from Q_1 only when the number of jobs in Q_1 has exceeded this threshold. More specifically, S_2 will give jobs in Q_1 higher priority over jobs in Q_2 (thus following the $c\mu$ -rule) only if there are more than T jobs in Q_1 ; otherwise, it serves jobs in Q_2 . Notice that this policy reduces to the $c\mu$ -rule when the threshold T is set to be 0.

Figure 2 shows for varying traffic intensities the corresponding long-run average inventory costs in this system under different threshold values. Observe that the threshold policy has the desired effect and that the best thresholds in all of these cases are quite small. When the traffic is light, the threshold policy and the $c\mu$ -rule yield very similar performance characteristics.

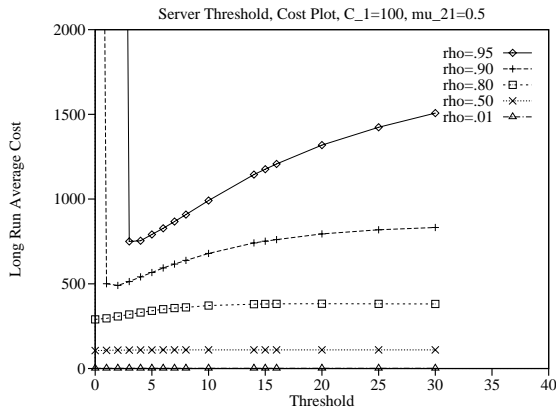


Figure 2: Inventory Cost for the Two-Server, Two-Queue Example

2.2 General Case

We now extend the idea of the above threshold-based priority policy to the general case with J job classes, M servers, and arbitrary μ_{ij} . For this purpose, it will be convenient to view the multiclass parallel-server system as a bipartite graph G , in which nodes represent either servers or queues, and there is an undirected link (i, j) between S_i and Q_j if $\mu_{ij} > 0$. Let Ω denote the set of all links in G .

The threshold-based policy can therefore be viewed as defining a subset \mathcal{T} of links (i, j) in Ω and setting a threshold T_{ij} for each link (i, j) in \mathcal{T} . When S_i becomes available, it first checks each Q_j , such that $(i, j) \in \mathcal{T}$, to determine whether the number of jobs in the queue has exceeded the corresponding threshold T_{ij} . If multiple queues satisfy this condition, S_i serves a job from one of these queues following the $c\mu$ rule. If none of the queues has exceeded its threshold value, S_i simply serves a job from one of the non-empty Q_j , such that $(i, j) \in \mathcal{T}$, following the $c\mu$ rule. When a C_j job arrives at an empty queue, it selects from among all idle S_i , such that $(i, j) \in \mathcal{T}$, the server with the largest rate μ_{ij} and immediately begins its service there. If no such idle servers exist, the job simply waits in Q_j .

Before proceeding with the technical details, several remarks are in order. First, the general threshold policy turns off link (i, j) if $(i, j) \notin \mathcal{T}$, i.e., S_i will remain idle even if $\mu_{ij} > 0$ and only Q_j has jobs waiting to be served. This makes it possible for us to include idling policies within our general threshold policy framework. Second, every link (i, j) in \mathcal{T} must have a threshold value even if the threshold $T_{ij} = 0$ (which is effectively *no* threshold). We therefore use $\mathcal{R} \subseteq \mathcal{T}$ to denote all links (i, j) in \mathcal{T} that have threshold values $T_{ij} > 0$. Note that \mathcal{T} and \mathcal{R} are different in that \mathcal{T} determines *where to help* whereas \mathcal{R} ensures that the help is *not too zealous*.

There are three key questions we have to answer to complete the characterization of our general threshold policy; namely, (i) how to choose \mathcal{T} , (ii) how to choose \mathcal{R} , and (iii) what are the optimal threshold values for each link in \mathcal{R} ? The remainder of this section addresses the first two issues, with the third issue studied in §4.

2.3 Where to Set the Thresholds

We observe that any policy which minimizes the long-run average inventory cost must maintain a stable system, assuming there exists a policy under which the system is stable. When system load is heavy, maintaining a stable system relates directly to allocating the capacity of the servers among the job classes so as to maximize system throughput. This optimal capacity allocation problem can be formulated as a linear program as follows. Suppose that the job arrival rates increase proportional to the initial rates λ_j , $j = 1, \dots, J$. Let x_{ij} be the proportion of capacity that S_i devotes to processing C_j jobs. The maximum load of the

M -server system can then be determined as

$$\begin{aligned}
\text{CA-LP} \quad & \max \rho \\
\text{s.t.} \quad & \sum_{i=1}^M \mu_{ij} x_{ij} = \lambda_j \rho, \quad j = 1, \dots, J; \\
& \sum_{j=1}^J x_{ij} \leq 1, \quad i = 1, \dots, M; \\
& x_{ij} \geq 0, \quad i = 1, \dots, M; \quad j = 1, \dots, J.
\end{aligned}$$

Let (x^*, ρ^*) be an optimal basic solution of this capacity allocation linear program CA-LP. Let \mathcal{T} be the set of links (i, j) that correspond to the basic variables in x^* . Specifically, a link (i, j) is in \mathcal{T} if and only if $x_{ij}^* > 0$. As described earlier, we will set thresholds for the links in \mathcal{T} .

Lemma 1 The graph \mathcal{T} must contain no cycles. Consequently, if the number of links in \mathcal{T} is exactly $J + M - 1$, then \mathcal{T} must be a spanning tree of the bipartite graph G . On the other hand, if \mathcal{T} has less than $J + M - 1$ links, then \mathcal{T} must contain more than one connected subgraphs, with each subgraph being a tree.

It follows from Lemma 1 that, if \mathcal{T} has less than $J + M - 1$ links, then the scheduling problem can simply be decomposed into multiple independent sub-problems. Henceforth, we suppose that the number of links in \mathcal{T} is $J + M - 1$.

The key is to identify those links where strictly positive thresholds must be set, lest the system becomes unstable. A good example is the diagonal link in the two-server, two-queue system of §2.1. Based on this example, we now define a *critical structure*, for the two-server, two-queue graph, that will serve as a building block for the general case.

Definition 2 For a two-server, two-queue graph with an “N” shape link structure (i.e., a connected graph with a tree structure as in Figure 1), in which i is the shared server and k is the shared queue, the graph is said to have the critical structure if $c_k \mu_{ik} > c_j \mu_{ij}$, where j is any other queue also served by \mathcal{S}_i .

The critical structure thus consists of those “N” shaped two-server, two-queue graphs in which the shared server favors the shared queue under the $c\mu$ rule. In light of the two-server, two-queue example, the diagonal link of such a critical structure should be controlled by a threshold so as to regulate the efforts of the shared server devoted to the shared queue. Recall that \mathcal{R} is the set of key links in \mathcal{T} for which non-zero thresholds are needed. Then the links in \mathcal{R} correspond to the diagonal links in the embedded critical structures of \mathcal{T} .

The algorithm below presents a systematic way to construct the set \mathcal{R} . It starts by scanning the leaves of the tree \mathcal{T} to identify an “N” shaped subgraph of \mathcal{T} . After comparing the different $c\mu$ values, it determines if the “N” shaped subgraph has the critical structure; if so, it then sets a threshold for the diagonal link. The algorithm then removes this leaf.

(0) Initialize: $\mathcal{R} = \emptyset$, $\mathcal{T}' = \mathcal{T}$.

(i) Pick an arbitrary leaf n from tree \mathcal{T}' .

Case 1: Leaf n is a queue. Identify among its servers a shared \mathcal{S}_i , i.e., link $(i, n) \in \mathcal{T}'$, that also serves other queues (e.g., see Figure 3(a)). Check all shared \mathcal{Q}_k such that $(i, k) \in \mathcal{T}'$ and k is not a leaf:

If $c_k \mu_{ik} \geq c_n \mu_{in}$, include (i, k) in \mathcal{R} .

Case 2: Leaf n is a server. Identify a shared \mathcal{Q}_k that is served by n , i.e., link $(n, k) \in \mathcal{T}'$, and k is also served by other servers (e.g., see Figure 3(b)). Check all shared \mathcal{S}_i such that $(i, k) \in \mathcal{T}'$ and i is not a leaf:

If there exists a \mathcal{Q}_j such that link $(i, j) \in \mathcal{T}'$ and $c_k \mu_{ik} \geq c_j \mu_{ij}$, include (i, k) in \mathcal{R} .

(ii) Remove leaf n from \mathcal{T}' and repeat (i) until at most two links are left in \mathcal{T}' .

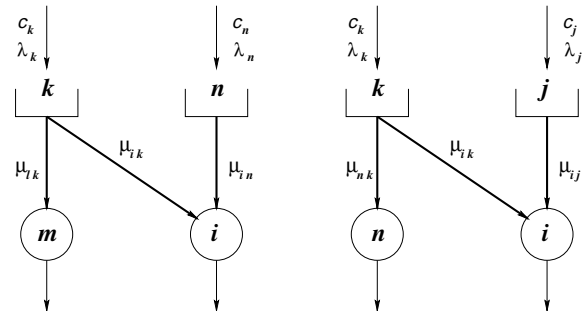


Figure 3: Illustrations for Case (1) and Case (2)

Notice that one can follow the above algorithm in many different ways, depending on which leaf was chosen in each iteration. However, it can be shown that the algorithm generates the same set \mathcal{R} independent of the order in which the leaves are selected in each iteration.

Proposition 3 \mathcal{R} consists of the diagonal link of each “N” shaped subset of \mathcal{T} that has the critical structure. Hence, \mathcal{R} is uniquely determined by the above algorithm.

3 Examples

We now present several examples to illustrate the use of our algorithm to determine the placement of thresholds, and to

quantitatively compare through simulation the long-run average inventory cost under the threshold-based priority policies and the $c\mu$ policy. Note that in all cases the simulations are run until the half-width of a 95% confidence interval is within 1% of the mean. For convenience, we use μ_i to denote the service rate vector for \mathcal{S}_i , and we use x_i^* to denote the vector of optimal fractions of service effort corresponding to \mathcal{S}_i .

Example 1. Consider the 3-server, 4-queue example illustrated in Figure 4, where the parameters are: $c = (1, 0.6, 2, 1)$, $\lambda = (0.5, 1, 2, 3)$, $\mu_1 = (1.25, 2.5, 1, 0)$, $\mu_2 = (0, 4, 2, 5)$, $\mu_3 = (0, 0, 0.8, 2.5)$. The optimal solution for the corresponding CA-LP is:

$$\begin{aligned} \rho^* &= 1; & x_1^* &= (0.4, 0.4, 0.2, 0); \\ x_2^* &= (0, 0, 0.9, 0.1); & x_3^* &= (0, 0, 0, 1). \end{aligned}$$

This then leads to the tree consisting of links $\mathcal{T} = \{(1, 1), (1, 2), (1, 3), (2, 3), (2, 4), (3, 4)\}$. The priorities for each server under the $c\mu$ rule are:

$$\mathcal{S}_1 : 3 > 2 > 1; \quad \mathcal{S}_2 : 4 > 3 > 2; \quad \mathcal{S}_3 : 4 > 3;$$

where $a > b$ means that \mathcal{C}_a has priority over \mathcal{C}_b . Our algorithm selects the set $\mathcal{R} = \{(1, 3), (2, 4)\}$ on which to place thresholds.

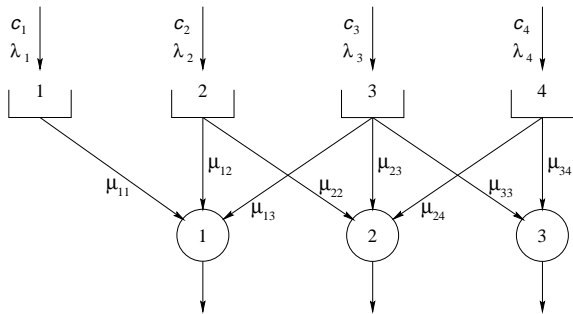


Figure 4: A Three-Server, Four-Queue Example

The uppermost plot of Figure 5 provides the results from simulation experiments of the corresponding system with traffic intensity $\rho = 0.95$ under the $c\mu$ -rule as well as the threshold policy with various threshold values. We observe that the system is unstable if the thresholds are 0, which corresponds to the $c\mu$ -rule. A set of fairly low threshold values for links (1, 3) and (2, 4) yields the best cost. Note that this is the same example as in [2] and Example 6.2 in [9]. The threshold policy given in [9] also determines to set thresholds for $\mathcal{R} = \{(1, 3), (2, 4)\}$.

We can also enable the links $\{(2, 2), (3, 3)\}$ such that \mathcal{T} contains the set of all links and \mathcal{R} remains the same. The lowermost plot in Figure 5 provides the long-run average cost obtained by simulation for this threshold policy. Observe that enabling the additional links does provide a slightly better cost. We note, however, that as ρ gets closer

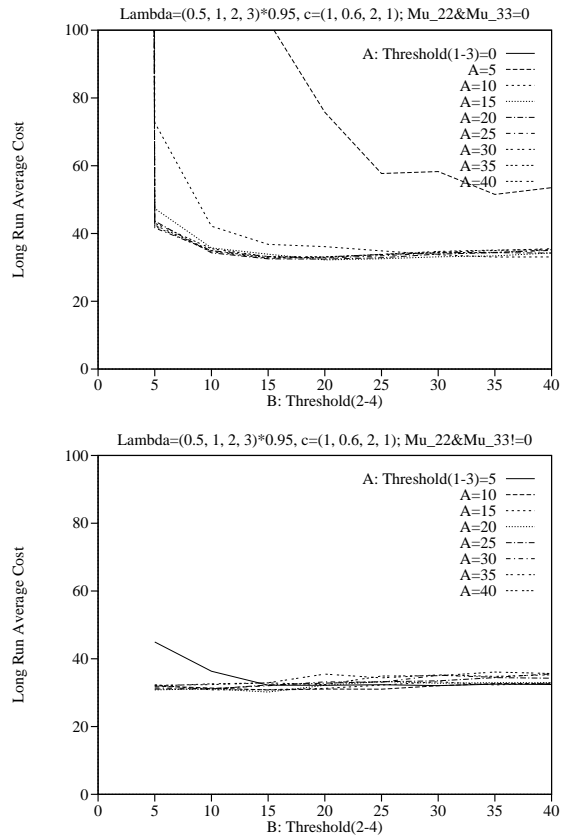


Figure 5: Long Run Average Cost for Example 1.

to 1, enabling the links $\{(2, 2), (3, 3)\}$ leads to an unstable system and thus provides a worse inventory cost, as demonstrated by additional simulation experiments.

Example 2. Consider the same 3-server, 4-queue system in Figure 4, but with a different set of parameters: $c = (3, 1, 0.5, 1)$, $\lambda = (1, 1, 3, 1)$, $\mu_1 = (1.25, 1, 5, 0)$, $\mu_2 = (0, 4, 2, 4)$, $\mu_3 = (0, 0, 1, 1)$. The optimal solution for the corresponding CA-LP is:

$$\begin{aligned} \rho^* &= 1; & x_1^* &= (0.8, 0, 0.2, 0); \\ x_2^* &= (0, 0.25, 0.5, 0.25); & x_3^* &= (0, 0, 1, 0). \end{aligned}$$

The tree \mathcal{T} has links $\{(1, 1), (1, 3), (2, 2), (2, 3), (2, 4), (3, 3)\}$. The priorities for each server under the $c\mu$ rule are:

$$\mathcal{S}_1 : 1 > 3 > 2; \quad \mathcal{S}_2 : 2 = 4 > 3; \quad \mathcal{S}_3 : 4 > 3.$$

The algorithm selects no links ($\mathcal{R} = \emptyset$) on which to set thresholds.

We note that this example is the same as Example 6.3 in [9]. The algorithm in [9] sets a threshold for link (2, 3). From the viewpoint of \mathcal{S}_2 , \mathcal{C}_3 jobs have the lowest priority among all of the job classes that \mathcal{S}_2 can process. Therefore, under the previously defined threshold policy, different values of the threshold for link (2, 3) do not change the behavior of the system. It is therefore not necessary to set a threshold on this link.

Example 3. Consider the 3-server, 3-queue system illustrated in Figure 6. The parameters are: $c = (1, 2, 1)$, $\lambda = (3, 2, 1)$, $\mu_{1\cdot} = (2.5, 0.2, 0)$, $\mu_{2\cdot} = (5, 2, 2.5)$, $\mu_{3\cdot} = (0, 1, 1.25)$. The corresponding CA-LP has two optimal basic solutions:

$$\begin{aligned} \rho^* &= 1; & x_{1\cdot}^* &= (1, 0, 0); \\ x_{2\cdot}^* &= (0.1, 0.9, 0); & x_{3\cdot}^* &= (0, 0.2, 0.8); \end{aligned}$$

and

$$\begin{aligned} \rho^* &= 1; & x_{1\cdot}^* &= (1, 0, 0); \\ x_{2\cdot}^* &= (0.1, 0.5, 0.4); & x_{3\cdot}^* &= (0, 1, 0). \end{aligned}$$

The tree \mathcal{T} has links $\{(1, 1), (2, 1), (2, 2), (3, 2), (3, 3)\}$ for the first solution, and $\{(1, 1), (2, 1), (2, 2), (2, 3), (3, 2)\}$ for the second solution. The priorities for each server under the $c\mu$ rule are:

$$\mathcal{S}_1 : 1 > 2; \quad \mathcal{S}_2 : 1 > 2 > 3; \quad \mathcal{S}_3 : 2 > 3 > 1.$$

Our algorithm then selects the following set of links on which to set thresholds: $\mathcal{R} = \{(2, 1), (3, 2)\}$ for the first solution; and $\mathcal{R} = \{(2, 1)\}$ for the second solution.

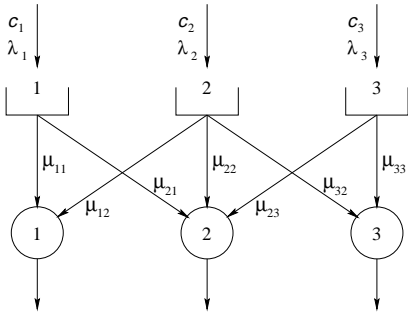


Figure 6: A Three-Server, Three-Queue Example

Various simulation results for this example are provided in Figure 7, where the leftmost plot illustrates the long-run average cost for the threshold policy that does not allow jobs to flow along links (1, 2) and (2, 3). Simulation experiments show that non-zero threshold values are needed for both links (2, 1) and (3, 2), and the best cost is achieved by setting both of these thresholds equal to 20. The middle plot of Figure 7 shows the inventory cost when jobs are allowed to flow along both links (1, 2) and (2, 3). In this case we observe that the best cost is achieved by setting the threshold on link (3, 2) equal to 0, and setting the threshold on link (2, 1) equal to 20.

The key reason for having the two optimal solutions is because $\mu_{23}\mu_{32} = \mu_{22}\mu_{33}$. This means that there is an even tradeoff between letting jobs flow along links $\{(2, 3), (3, 2)\}$ and along links $\{(2, 2), (3, 3)\}$. Hence, the threshold on link (3, 2) is not necessary. The rightmost plot of Figure 7 shows the long-run average cost for the threshold policy that sets thresholds on links (2, 1) and (3, 2), and

allows jobs to flow along link (2, 3) but not along (1, 2). Simulation experiments show that the best cost is always achieved by setting the threshold on link (3, 2) equal to 0. Furthermore, non-zero threshold values are always needed on link (2, 1), and the best cost is achieved by setting this threshold equal to 10. Note that in this case the inventory cost is the best among all three settings considered in our experiments.

Example 4. Consider the 3-server, 3-queue system in Figure 8. The parameters are: $c = (10, 100, 1)$, $\lambda = (0.4, 1.4, 0.6)$, $\mu_{1\cdot} = (1, 0.5, 0)$, $\mu_{2\cdot} = (0, 1, 0)$, $\mu_{3\cdot} = (0, 0.25, 1)$. The optimal solution for the corresponding CA-LP is:

$$\begin{aligned} \rho^* &= 1; & x_{1\cdot}^* &= (0.4, 0.6, 0); \\ x_{2\cdot}^* &= (0, 1, 0); & x_{3\cdot}^* &= (0, 0.4, 0.6). \end{aligned}$$

The tree consists of all of the links $\mathcal{T} = \{(1, 1), (1, 2), (2, 2), (3, 2), (3, 3)\}$, since the graph itself is a tree. The priorities for each server under the $c\mu$ rule is:

$$\mathcal{S}_1 : 2 > 1; \quad \mathcal{S}_2 : 2 = 2; \quad \mathcal{S}_3 : 2 > 3.$$

Our algorithm selects the set of links $\mathcal{R} = \{(1, 2), (3, 2)\}$ on which to set thresholds.

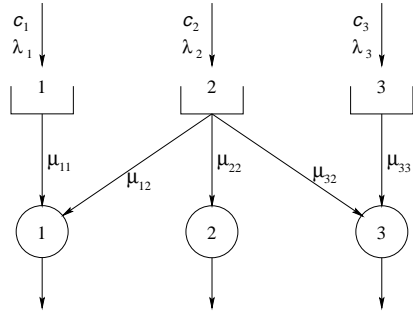


Figure 8: A Three-Server, Three-Queue Example

The two plots in Figure 9 provide the long-run average cost for the threshold policy with various threshold values on links (1, 2) and (3, 2). We observe that non-zero threshold values are needed for both links to avoid system instability. Moreover, the best cost is achieved by setting both of the threshold values equal to 10.

Example 5. Consider the 3-server, 3-queue system illustrated in Figure 10 with the parameters: $c = (100, 1, 10)$, $\lambda = (1.1, 0.5, 1.24)$, $\mu_{1\cdot} = (1, 0, 0)$, $\mu_{2\cdot} = (0.5, 1, 0.8)$, $\mu_{3\cdot} = (0, 0, 1)$. The optimal solution for the corresponding CA-LP is:

$$\begin{aligned} \rho^* &= 1; & x_{1\cdot}^* &= (1, 0, 0); \\ x_{2\cdot}^* &= (0.2, 0.5, 0.3); & x_{3\cdot}^* &= (0, 0, 1). \end{aligned}$$

The tree \mathcal{T} contains all of the links $\{(1, 1), (2, 1), (2, 2), (2, 3), (3, 3)\}$, since the graph

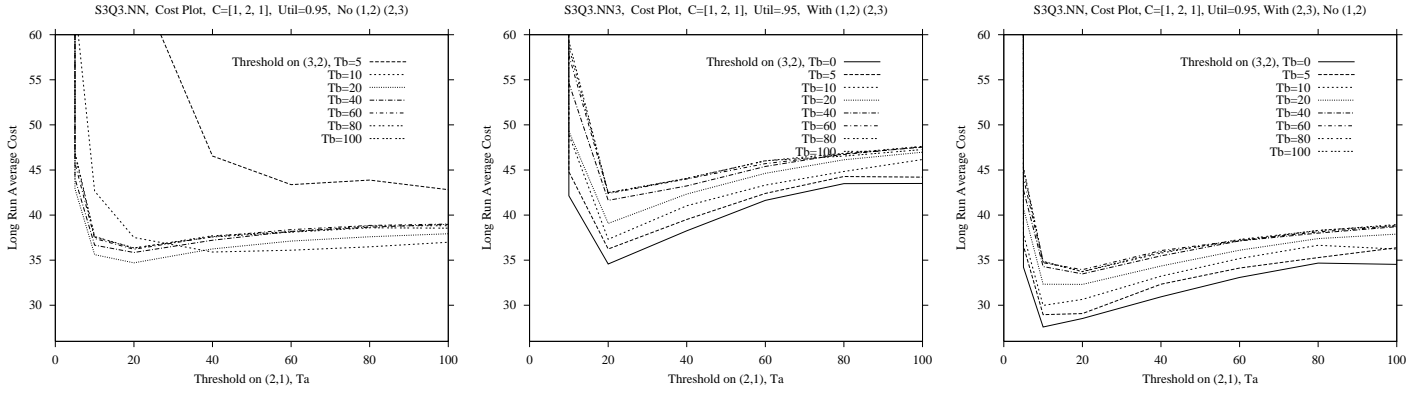


Figure 7: Long Run Average Cost for Example 3.

itself is a tree. The priorities for each server under the $c\mu$ rule is:

$$\mathcal{S}_1 : 1 = 1; \quad \mathcal{S}_2 : 1 > 3 > 2; \quad \mathcal{S}_3 : 3 = 3.$$

Our algorithm selects the set of links $\mathcal{R} = \{(2, 1), (2, 3)\}$ on which to set thresholds.

The two plots in Figure 11 provide the long-run average cost for the threshold policy with various threshold values on links (2, 1) and (2, 3). We observe that non-zero threshold values are needed for both links to avoid an unstable system, and the best cost is achieved by setting both of the threshold values equal to 10.

4 Approximations

In this section we derive closed-form approximate formulas for the queue lengths under the threshold-based priority rules discussed in the previous sections, and then we illustrate how these formulas can be used to derive optimal threshold values. To facilitate the exposition, our focus is on the two-queue two-server case. The results, however, extend readily to more general cases.

Consider the example in Figure 1. Recall that there are two independent Poisson arrival streams, with rates λ_1 and λ_2 , and there are two exponential servers, with \mathcal{S}_1 processing \mathcal{C}_1 jobs at rate μ_{11} and \mathcal{S}_2 processing \mathcal{C}_2 jobs at rate μ_{22} . \mathcal{S}_2 can also process \mathcal{C}_1 jobs, but at a slower rate $\mu_{21} \leq \mu_{22}$. Furthermore, we assume that $\mu_{21} \leq \mu_{11}$, $c_1\mu_{21} \geq c_2\mu_{22}$, and $\mu_{12} = 0$.

While \mathcal{S}_1 by itself does not have enough capacity to process all \mathcal{C}_1 jobs, \mathcal{S}_2 has excess capacity for \mathcal{C}_2 jobs such that between the two servers there is enough capacity to handle both job classes. Following the analysis of §2, under the above assumptions, we employ a threshold T at \mathcal{Q}_1 . Specifically, \mathcal{S}_2 will process jobs from \mathcal{Q}_1 whenever \mathcal{Q}_2 is empty, or whenever \mathcal{Q}_2 is non-empty *and* there are more than T jobs in \mathcal{Q}_1 . When \mathcal{Q}_2 is non-empty and there are T or fewer jobs in \mathcal{Q}_1 , \mathcal{S}_2 will process jobs from \mathcal{Q}_2 .

To model this threshold policy analytically, we shall approximate \mathcal{Q}_1 as a birth-death process having death rate $\mu_{11} + \mu_{21}(1 - r_2)$ when the queue has T or fewer jobs, and having death rate $\mu_{11} + \mu_{21}$ when the queue has more than T jobs. Here, r_2 represents the proportion of time that \mathcal{Q}_2 is non-empty. Let $\{\pi_i\}$ denote the stationary distribution of the number of jobs in this birth-death process, and define

$$\rho_{\leq T} = \frac{\lambda_1}{\mu_{11} + \mu_{21}(1 - r_2)}, \quad \rho_{> T} = \frac{\lambda_1}{\mu_{11} + \mu_{21}}. \quad (1)$$

We then have

$$\pi_i = \begin{cases} \pi_0 \rho_{\leq T}^i, & i \leq T, \\ \pi_0 \rho_{\leq T}^T \rho_{> T}^{i-T}, & i > T, \end{cases}$$

where

$$\pi_0 = \left[\frac{1 - \rho_{\leq T}^{T+1}}{1 - \rho_{\leq T}} + \frac{\rho_{\leq T}^T \rho_{> T}}{1 - \rho_{> T}} \right]^{-1}. \quad (2)$$

From this we obtain

$$L_1 = \pi_0 \left[\frac{\rho_{\leq T} - (T+1)\rho_{\leq T}^{T+1} + T\rho_{\leq T}^{T+2}}{(1 - \rho_{\leq T})^2} + \rho_{\leq T}^T \frac{(T+1)\rho_{> T} - T\rho_{> T}^2}{(1 - \rho_{> T})^2} \right]. \quad (3)$$

To approximate \mathcal{Q}_2 , on the other hand, we use an M/G/1 queue with server vacation, where the vacations represent the time spent by \mathcal{S}_2 in processing jobs from \mathcal{Q}_1 . Specifically, we model \mathcal{S}_2 as following a Bernoulli vacation mechanism in which it goes on vacation with probability

$$p_v := \sum_{j=1}^{\infty} \pi_{T+j} = \frac{\pi_0 \rho_{\leq T}^T \rho_{> T}}{1 - \rho_{> T}}. \quad (4)$$

The vacation period, denoted V , is given by

$$V := S_{21}^1 + S_{21}^2 + \cdots + S_{21}^N,$$

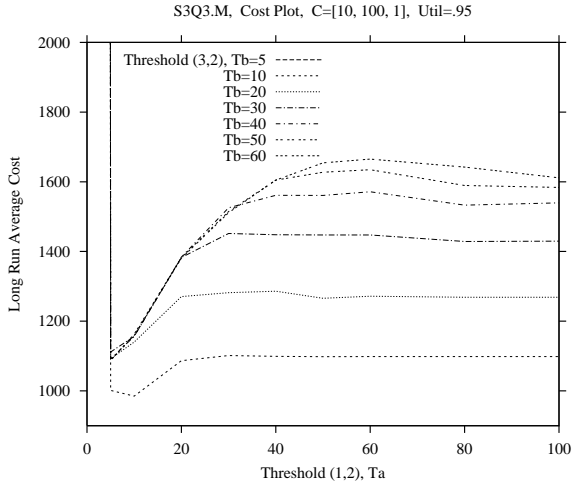
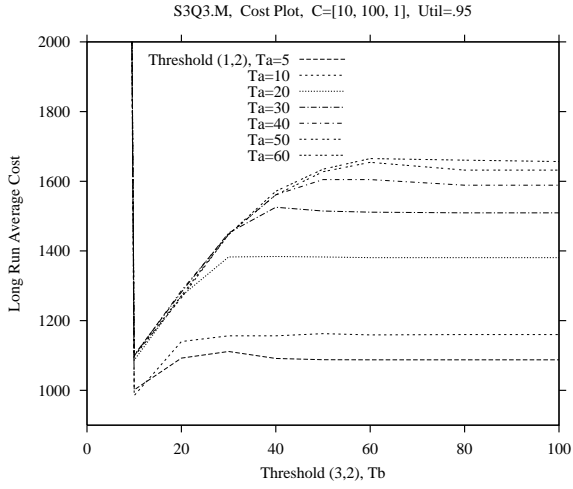


Figure 9: Long Run Average Cost for Example 4.

where $\{S_{21}^i; i = 1, 2, \dots\}$ is a sequence of independent and identically distributed exponential service times with mean μ_{21}^{-1} , and N follows a geometric distribution:

$$P[N = n] = (1 - p_v)p_v^n, \quad n = 1, 2, \dots$$

Thus, we have

$$r_2 = \frac{\lambda}{\mu_{22}(1 - p_v)}. \quad (5)$$

Note that

$$E[N] = \frac{1}{1 - p_v}, \quad \text{Var}[N] = \frac{p_v}{(1 - p_v)^2};$$

hence,

$$E[V] = E[N]E[S_{21}] = \frac{1}{\mu_{21}(1 - p_v)}, \quad (6)$$

and

$$\begin{aligned} \text{Var}[V] &= E[N]\text{Var}[S_{21}] + \text{Var}[N]E^2[S_{21}] \\ &= \frac{1}{\mu_{21}^2(1 - p_v)^2} = E^2[V]. \end{aligned} \quad (7)$$

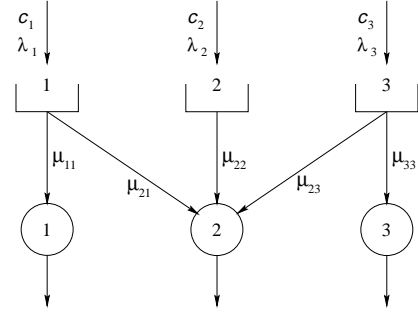


Figure 10: A Three-Server, Three-Queue Example

Following standard results in server-vacation models (e.g., [5]), we have

$$L_2 = \lambda_2 \{E[W(S_2 + Y)] + E[R_V]\} + \rho_2, \quad (8)$$

where

$$E[R_V] = \frac{E[V^2]}{2E[V]} = \frac{\text{Var}[V] + E^2[V]}{2E[V]} = \frac{2E^2[V]}{2E[V]} = E[V],$$

$\rho_2 = \lambda_2/\mu_{22}$, and $W(S_2 + Y)$ denotes the waiting (queueing) time in a regular M/G/1 model with service times represented by $S_2 + Y$, where S_2 and Y are two independent random variables, with S_2 denoting the original service times (at S_2 , i.e., exponential with mean μ_{22}^{-1}), and $Y = V$ with probability p_v and zero otherwise. We then have

$$E[Y] = p_v E[V] = \frac{p_v}{\mu_{21}(1 - p_v)}$$

and

$$E[Y^2] = p_v E[V^2] = \frac{2p_v}{\mu_{21}^2(1 - p_v)^2} = \frac{2}{p_v} E^2[Y],$$

where the second equality follows from (6) and (7).

Following the standard M/G/1 result, we have

$$E[W(S_2 + Y)] = \frac{\lambda_2 E[(S_2 + Y)^2]}{2[1 - \lambda_2 E(S_2 + Y)]} := \frac{W_0^V}{1 - \rho_2^V}, \quad (9)$$

where

$$\rho_2^V := \lambda_2 E(S_2 + Y) = \rho_2 + \frac{\lambda_2 p_v}{\mu_{21}(1 - p_v)},$$

and

$$\begin{aligned} W_0^V &:= \frac{\lambda_2}{2} E[(S_2 + Y)^2] \\ &= \frac{\lambda_2}{2} \left[\frac{2}{\mu_{22}^2} + \frac{2}{\mu_{22}} E(Y) + E(Y^2) \right] \\ &= \lambda_2 \left[\frac{1}{\mu_{22}^2} + \frac{p_v}{\mu_{22}\mu_{21}(1 - p_v)} + \frac{p_v}{\mu_{21}^2(1 - p_v)^2} \right]. \end{aligned}$$

We are now ready to determine the optimal threshold that minimizes the long-run average holding cost. Let L_i^T represent the long-run average C_i queue length under the threshold policy with threshold T . It suffices to solve

$$\min_T [c_1 L_1^T + c_2 L_2^T], \quad (10)$$

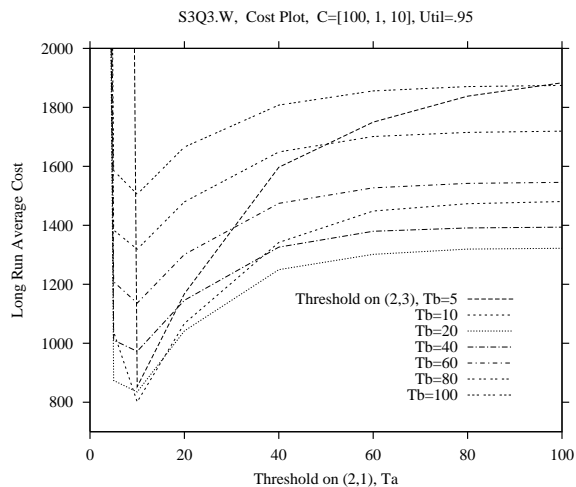
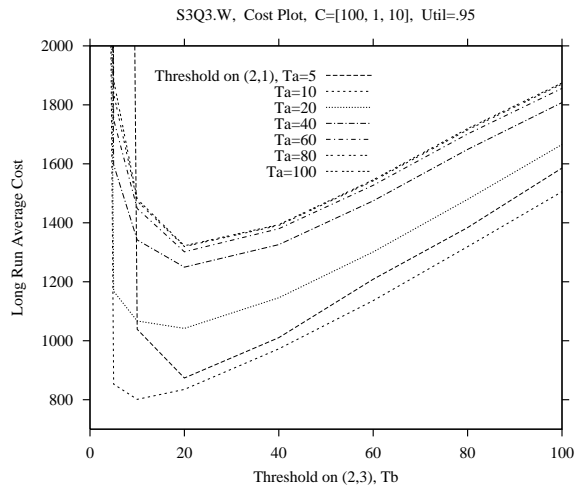


Figure 11: Long Run Average Cost for Example 5.

for the optimal threshold T^* . Here we approximate L_1^T and L_2^T using (3) and (8), respectively.

In Figure 12, we plot the long-run average cost obtained from the above approximate formulas for the parallel-server system in Figure 1 under different threshold values, different traffic intensities, and the parameters given in §2.1. For comparison, Figure 12 also provides the results from the corresponding simulation experiments of the same parallel-server threshold policy. Observe that the optimal threshold value obtained from solving (10) compares favorably with the best choice of threshold determined empirically from simulation results. In particular, our scheme works extremely well in light traffic, while in heavy traffic the approximation method tends to over-estimate somewhat the optimal threshold value. We note, however, that the long-run average cost deteriorates sharply for values of T below the optimum, but does so much more slowly for thresholds above the optimal value. This indicates that minor over-estimation does not hurt the inventory cost too much, and it is the under-estimation of the threshold value that should be avoided.

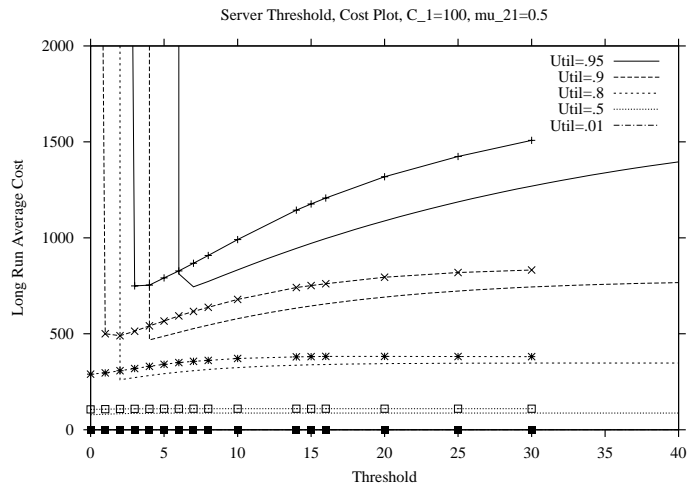


Figure 12: Approximation (pure-lines) v.s. Simulation (lines with points)

References

- [1] HARRISON, J.M., Heavy traffic analysis of a system with parallel servers: Asymptotic optimality of discrete-review policies. *Ann. Appl. Prob.*, **8**:822-848, 1998.
- [2] HARRISON, J.M., LOPÉZ, M.J., Heavy traffic resource pooling in parallel-server systems. *Queueing Systems*, to appear.
- [3] KLEINROCK, L., *Queueing Systems*, Vol. II, Wiley Interscience, New York, 1976.
- [4] LAWS, C.N., Resource pooling in queueing networks with dynamic routing. *Adv. Appl. Prob.*, **24**:699-726, 1992.
- [5] SERVI, L.D., YAO, D.D., Stochastic bounds for queueing systems with limited service schedules. *Perf. Eval.*, **9**:247-261, 1988.
- [6] SQUILLANTE, M.S., LAZOWSKA, E.D., Using processor-cache affinity information in shared-memory multiprocessor scheduling. *IEEE Trans. Parallel and Dist. Systems*, **4**:131-143, 1993.
- [7] SQUILLANTE, M.S., NELSON, R.D., Analysis of task migration in shared-memory multiprocessors. *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, 143-155, 1991.
- [8] SQUILLANTE, M.S., XIA, C.H., YAO, D.D., ZHANG, L., Optimal control of a multiclass parallel-server fluid network. In preparation.
- [9] WILLIAMS, R.J., On dynamic scheduling of a parallel server system with complete resource pooling. *To appear*.