J. Appl. Prob. **48,** 435–452 (2011)
Printed in England
© *Applied Probability Trust* 2011

MANAGING QUEUES WITH HETEROGENEOUS SERVERS

JUNG HYUN KIM,[*] *Korea Telecommunication Corporation*
HYUN-SOO AHN,[**] *University of Michigan*
RHONDA RIGHTER,[***] *University of California, Berkeley*

Abstract

We consider several versions of the job assignment problem for an M/M/m queue with servers of different speeds. When there are two classes of customers, primary and secondary, the number of secondary customers is infinite, and idling is not permitted, we develop an intuitive proof that the optimal policy that minimizes the mean waiting time has a threshold structure. That is, for each server, there is a server-dependent threshold such that a primary customer will be assigned to that server if and only if the queue length of primary customers meets or exceeds the threshold. Our key argument can be generalized to extend the structural result to models with impatient customers, discounted waiting time, batch arrivals and services, geometrically distributed service times, and a random environment. We show how to compute the optimal thresholds, and study the impact of heterogeneity in server speeds on mean waiting times. We also apply the same machinery to the classical slow-server problem without secondary customers, and obtain more general results for the two-server case and strengthen existing results for more than two servers.

Keywords: Multiserver queue; heterogeneous server; individual and social optimality

2010 Mathematics Subject Classification: Primary 60K25; 60K30

1. Introduction

In this paper we consider a job assignment problem in a queueing system with multiple servers of different service rates. The optimal job assignment policy to minimize average waiting time when there are only two heterogeneous servers and identical jobs has been extensively studied with different approaches. It has been shown that the optimal policy is a threshold type, that is, the faster server is always used and a job should be assigned to the slower server if and only if the number of jobs in the queue exceeds the threshold level. Lin and Kumar [7] were the first to show these properties of the optimal policy, using a dynamic programming approach. Koole [5] later provided a simplified dynamic programming proof of the optimality of a threshold on the queue length for use of the slow server, exploiting the separately proved fact that the fastest server should always be used. Walrand [17] showed the two-server threshold result with a sample path argument. Xu [18] used a different approach. She first considered the individually optimal policy in which each job tries to minimize its own

Received 27 April 2009; revision received 7 June 2010.
[*] Email address: javenue@gmail.com
[**] Postal address: Ross School of Business, University of Michigan, 701 Tappan Street, Ann Arbor, MI 48109-1234, USA. Email address: hsahn@umich.edu
[***] Postal address: Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720, USA. Email address: rrighter@ieor.berkeley.edu

waiting time, under the assumption that more recent arrivals have priority in choosing servers, and preemption is allowed (the LCFP-P, last-come–first-priority with preemption, discipline). She then showed that the individually optimal policy under LCFP-P is a threshold type, and it is also socially optimal. Note that the socially optimal policy minimizes the average waiting time experienced by the system and that it does not matter which jobs are assigned as jobs are identical. Stockbridge [15] used a martingale approach to reduce the problem to a linear programming problem. He also showed that in a finite buffer system, with the objective of minimizing the average waiting time of served jobs, the optimal policy may not be a threshold policy; when the queue is nearly full, it may be better not to use the slow server so that new jobs will be lost and experience no waiting.

The problem with more than two servers, however, turns out to be much more difficult, and the approaches that can be used in the two-server case fail to apply. It seems reasonable that the optimal policy would be a state-dependent threshold policy. That is, for a given state of busy and idle servers, there is a threshold associated with that state, such that if the number of jobs in the queue exceeds the threshold, a job should be assigned to the fastest available server. So far, there is no intuitive, probabilistic proof of this result, except in the case when there are no arrivals. Agrawala *et al.* [1] showed that with no arrivals the optimal policy is a threshold type and the decision to use a server is independent of the states of the slower servers. Kumar and Walrand [6] showed that the decisions made under the (socially) optimal policy described in Agrawala *et al.* [1] are optimal for each job individually as well. Righter [9] and Righter and Xu [10] considered heterogeneous jobs and more general cost functions, such as expected weighted flow time, weighted discounted flow time, and weighted number of tardy jobs. They also considered service times with nondecreasing hazard rates, where hazard rates for different servers are nonoverlapping [11]. They showed that if jobs with higher weights are given higher priorities then the individually optimal policy is also the socially optimal policy.

There have been several papers showing the partial structure of the socially optimal policy for models with arrivals. Rosberg and Makowski [12] showed that if the arrival rate is small enough, i.e. for $\lambda \leq \lambda_0$ for some λ_0, the optimal policy is the same as that for the no-arrival ($\lambda = 0$) case. For general arrival rates, the structure of the socially optimal policy is less clear. Weber [16] provided a three-server example showing that if a threshold policy is optimal, the threshold for using a server (e.g. server 2) would generally depend on the states of slower servers (e.g. server 3). Moreover, Weber's example shows how difficult it is to identify the structure of the socially optimal policy. In fact, under the socially optimal policy, there are sample paths such that at one point in time there is a single job in the queue that is not assigned to an available server under the socially optimal policy, yet later that same job will be assigned to that server. Kumar and Walrand [6] showed that if a job never uses a server that it previously declined to use under the socially optimal policy, then that policy is also individually optimal. However, they did not show that there exists such a socially optimal policy in the model with arrivals. Indeed, from Weber's example, we know that the socially optimal policy does not have that property. Rosberg and Makowski [12] and Weber [16] showed, using a sample path argument, that, for the M/M/n heterogeneous server system ($n \geq 1$), whenever the optimal policy assigns a job to an available server, it uses the fastest available server, and the very fastest server is always used. Weber conjectured that a threshold policy is optimal, and the threshold should increase as more (slower) servers became available. More recently, two approaches have been attempted to prove the optimality of a threshold policy. Rykov [13] used a dynamic programming value iteration approach, but de Vericourt and Zhou [4] have shown that the proof is incomplete. Luh and Viniotis [8] used a linear programming approach and sample path analysis to show that the optimal policy has a threshold structure, but their arguments are nonintuitive. Armony and Ward [3] considered the problem where there is a fairness constraint on the division of the workload among fast and slow servers, and Armony and Mandelbaum [2] considered optimal staffing levels for heterogeneous server systems in heavy traffic.

We first consider a variant of the problem in which there are two classes of jobs, which we call primary and secondary jobs. Primary jobs have priority and arrive according to a Poisson process. There are an infinite number of secondary jobs, and if at any time an available server is not used by a primary job, a secondary job will be assigned to it. An application is to blended call centers, in which incoming calls have priority, but when the queue is empty, idle servers make outbound calls (e.g. to generate sales or follow up earlier calls). The objective is to minimize the mean waiting time for primary jobs when preemption is not permitted. For this model, we show that the optimal policy is defined by a set of thresholds, one for each server, such that a primary job is assigned to the fastest available server if the number of primary jobs exceeds the threshold for that server. Moreover, the thresholds are decreasing in the speed of the server. We note that our results do not depend on assuming a specific service discipline for determining which job is assigned to the server; thus, our results hold under widely used disciplines such as first-come–first-served (FCFS), last-come–first-served (LCFS), random order, etc. To prove the optimality of a threshold policy, a typical approach is to write out a standard dynamic program for a discounted version of the problem, use an induction argument based on value iteration, and then generalize to the average cost problem by applying standard arguments. However, such a proof is tedious, provides little intuition and is difficult to generalize. We use an alternative approach, based on Xu's arguments [18]. We first study an individually optimal policy for primary jobs, where priority is given to jobs in reverse order of their arrival times. This priority forces jobs to consider future arrivals in making their decisions. The resulting individually optimal policy is almost immediately seen to be a threshold policy. We then show that, even when we remove the LCFS priority, this policy is socially optimal since any deviation from it makes all primary jobs worse off. Our proof is short and intuitive, and easily generalizes to models with batch arrivals, batch services, impatient customers, discounting, general service time and interarrival time distributions, and random environments. Also, by considering the individually optimal policy, we show how to compute the optimal thresholds. We show for the case of two servers that generally waiting times under the optimal policy are smaller when the speeds are less balanced, holding the sum of the speeds constant. Also, there needs to be a fairly large imbalance in the speeds for it to be optimal to not use the slow machine when primary jobs are waiting, especially in heavy traffic. For example, when $\rho = 0.9$, the fast server must be eleven times faster than the slow server in order to avoid using the slow server when customers are waiting.

We apply the same machinery to the original problem without secondary jobs, and we obtain more general results for the two-server case, and strengthen the known partial structural results (see [12] and [16]) for more than two servers. We also give structural results for the individually optimal policy under LCFP-P for two or more servers. Unlike the problem with secondary jobs, the individually optimal (IO) policy under LCFP-P is no longer socially optimal for more than two servers. We explain why our intuitive argument and other sample-path-based arguments (such as Walrand's [17] and Xu's [18], both of which work for the two server case) fail to work with more than two servers. Providing an intuitive proof for the complete structure of the optimal policy for the original job assignment problem remains as future work.

2. Model with two classes of jobs

2.1. Basic result

The system consists of one queue served by m servers, and the service rate of server i is μ_i. The servers operate in parallel, and service times are independent and exponentially distributed. Without loss of generality, suppose that server i is the ith fastest, i.e. $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_m$. The primary jobs arrive according to a Poisson process with rate λ and have priority over secondary jobs. We assume that $\lambda < \sum \mu_i$ for stability. Suppose that we have an infinite supply of 'dummy', or secondary, jobs that are always assigned to a server that is not used by any primary job. No jobs, including secondary jobs, can be preempted. Our objective is to minimize the mean waiting time (in the queue plus in service) for primary jobs, or, equivalently, from Little's law, the mean number of primary jobs in the system. With this objective, the service discipline (deciding which primary job to assign to a server) is irrelevant, though generally it will be FCFS in applications.

An interesting application of this model is a blended call center in which servers that are not busy handling in-bound calls (e.g. calls from customers) make out-bound calls (e.g. sales calls to customers). In this example, the in-bound calls are the primary jobs. We can also think of our model as a vacation model in which servers that do not start serving a (primary) job go on a vacation (i.e. serve a secondary job) upon a service completion or vacation return. A model in which low-priority jobs are processed instead of idling servers is appropriate in situations when there is plenty of less profitable work to be done, and where that work still generates net benefits for the company. Although adding secondary jobs makes the topology of the queueing system more complex, it turns out that this simplifies the analysis because there will be at most one server available at any given time.

We first consider the individually optimal policy, assuming that jobs are offered available servers according to the LCFP (last-come–first-priority) discipline. That is, at any time, primary jobs in the queue are numbered in descending order of their arrival times, and a primary job with a lower index possesses a higher priority to access servers, but cannot preempt jobs already at servers. The IO policy is a myopic policy under which each job tries to minimize its own expected waiting time. Therefore, a job will reject an available server i if and only if its expected waiting time from using the server, $1/\mu_i$, is strictly larger than waiting for a faster server to be available, where we assume without loss of generality that the server is accepted in the case of indifference. Note that LCFP captures the negative externalities that jobs impose on other jobs by giving higher priority to the jobs yet to come than to the jobs already in the system. Therefore, it forces present jobs to consider future jobs when they pursue their individual optimum. Also, note that, under the LCFP-IO policy, it is possible for a job to accept a server that it previously rejected. We show that the LCFP-IO policy is a threshold type, and it is socially optimal (SO) as well. That is, the IO policy also minimizes the mean waiting time for the whole system. Note that because job identities do not matter for our (social) objective function, the SO policy can also be implemented by using the IO thresholds, but assigning any primary job to the available server when that server's threshold is exceeded, not necessarily the job that would accept it under LCFP-IO. In particular, a possible SO policy is to assign jobs in FCFS order, but using the IO thresholds.

Since a primary job needs to consider only primary jobs that arrived, or will arrive, later than itself when deciding its IO policy, and the servers are busy with secondary jobs if no primary jobs are using them (also, no preemption is allowed), the decision for any job is indifferent to which jobs are at which servers. Therefore, without loss of generality, we remove the index of a primary job once it is assigned to a server, and assign indices to the jobs in the queue in descending order of their arrival times. There will be at most one server available at any instant, so our state at a decision time is the available server (which just finished a job) and the number of jobs in the queue (primary jobs).

We have the following result, where here and throughout we use the terms increasing and decreasing in the nonstrict sense. Let v_k be the expected waiting time in the system for job k under the LCFP-IO policy when all servers are busy.

Theorem 1. *The individually optimal policy under LCFP has the following properties.*

(i) *There exist thresholds $0 < T_j < \infty$ such that if server j is available and the index of a job is at least T_j, it is individually optimal for the job to use server j, i.e. job T_j will use server j if there are at least T_j jobs in the queue. Otherwise, waiting is optimal.*

(ii) $T_1 = 1$ *(so the fastest server is always used) and the threshold T_j is increasing in j.*

(iii) *The expected waiting time in the system for job k when all servers are busy, v_k, is increasing in k.*

Proof. (i) Fix j, and suppose that server j is available. If job k is offered the server and accepts it, its mean waiting time is $1/\mu_j$. Otherwise, because some other job will be assigned to j, the mean waiting time for job k will be v_k. Therefore, the IO policy for job k, assuming that jobs $1, \dots, k-1$ have rejected server j, is to accept it if $1/\mu_j \le v_k$. Thus, the IO policy is to assign job T_j to server j, where

$$T_j = \arg\min\left\{k : \frac{1}{\mu_j} \le v_k\right\}. \tag{1}$$

To show that $T_j < \infty$, note that $v_k \ge k/m\mu_1$, which would be the IO waiting time if all servers had service rate equal to μ_1, and there were no new arrivals.

(ii) We have $T_1 = 1$ because $v_1 > 1/\mu_1$; v_1 is the sum of the time to assign job 1 to a server (which is greater than 0 because we must wait for a server) and the time to serve the job (which is greater than or equal to $1/\mu_1$). From (1), T_j is increasing in j.

(iii) Because jobs are optimizing their own waiting times, and they are offered servers in increasing order of their index, jobs with higher indices (lower priorities) have longer expected waiting times.

We now show that the LCFP-IO policy is also SO, i.e. it minimizes the mean waiting time for all jobs.

Theorem 2. *The LCFP-IO policy is an SO policy; therefore, the SO policy is a threshold type where the thresholds correspond to those under LCFP-IO, and the threshold function T_j increases in j, with $T_1 = 1$.*

Proof. Let π^{IO} and π^{SO} respectively be the LCFP-IO and SO policies. The result is shown by policy iteration. Let π' be the policy that tries to improve upon π^{IO} by taking an action that is different from that of π^{IO} at the first decision epoch (time 0), and that follows the IO policy thereafter. We prove that such a π' is no better (socially) for any initial state, and, therefore, $\pi^{\mathrm{IO}} = \pi^{\mathrm{SO}}$. In particular, we show that if $\pi' \ne \pi^{\mathrm{IO}}$ for at least one initial state then we can assign identities to the jobs such that the expected waiting time for *each* job under π' will be at least as large as under π^{IO}.

Assume that the (say n) jobs in the queue at time 0 are indexed according to LCFP. First suppose that π' does not use a server for a primary job that would have been used by job $k \le n$ under π^{IO}, i.e. π' assigns a secondary job to the server. Job k is clearly better off under π^{IO} than under π', by the definition of the IO policy. Under π^{IO}, immediately after the job assignment at time 0, the indices of the jobs that have lower priority than job k are decrease by 1. That is, job l under π' corresponds to job $l-1$ under π^{IO}, $l = k+1, \dots, n$, where n is the number of jobs in the queue initially. Because both policies will follow the IO policy from then on, and because v_l is increasing in l, jobs $k+1, \dots, n$ will also be better off under π^{IO} than under π'. Finally, jobs $1, \dots, k-1$ and all future arrivals will have identical waiting times under both policies.

Now consider the case where π' makes a primary job use a server that would have been used by a secondary job under π^{IO}. As any job can be assigned, let π' assign the highest indexed job (under LCFP) in the queue, job n. Again, this decision has no impact on the dynamics of lower indexed jobs (higher priority jobs) in the queue or future arrivals, so all jobs except job n will have the same waiting time under π' and π^{IO}. As for job n, using the server is not IO for it, so it is worse off under π'. Again, the waiting times for all jobs are at least as long under π' as under π^{IO}. Therefore, the LCFP-IO policy is SO, and satisfies Theorem 1(i) and (ii).

The following corollary provides a light-traffic result for the model with secondary jobs that is similar to that of Rosberg and Makowski [12] for the original model with only one type of job. Its proof follows from (1) and the fact that v_k is increasing in λ. Here we make the dependency on λ explicit.

Corollary 1. *It holds that $T_j(\lambda)$ is decreasing in λ.*

A direct consequence of the corollary is that $T_j(0) \ge T_j(\lambda)$ for any λ. For $\lambda = 0$ and no secondary jobs, the optimal policy is to assign a job to the fastest available server, say j, if $k \ge R_j$, where

$$R_j := \left\lceil \frac{\mu_1 + \mu_2 + \cdots + \mu_{j-1}}{\mu_j} - (j-1) \right\rceil,$$

and if a server is rejected, it is never used again [1]. Therefore, the same policy can be implemented even with secondary jobs, so must still be optimal in that case. This gives us easily computable bounds for the optimal thresholds. For completeness, and to help develop intuition, we derive the optimal policy when $\lambda = 0$.

First note that with no arrivals, once a server is rejected, it, and all slower servers, will never be used again. Let v_k^i be the expected waiting time in the system for job k under the optimal policy with no arrivals, and assume that the only servers are servers $1, \dots, i$ and that they are all busy. Therefore, $v_k^1 = (k+1)/\mu_1$. Also, if job k is offered server j, it will use it if $1/\mu_j \le v_k^{j-1}$, where v_k^{j-1} is increasing in k. Therefore, we will have $T_j(0) = R_j$ as given above as long as $v_k^i = (k+i)/\sum_{l=1}^{i} \mu_l$ for $k \ge R_i$. We already have this for $i = 1$, so suppose that $i \ge 2$. First consider $k = R_i$. Then,

$$v_k^i = \frac{1}{\sum_{l=1}^{i} \mu_l} + \frac{\sum_{l=1}^{i-1} \mu_l}{\sum_{l=1}^{i} \mu_l} v_{k-1}^{i-1} + \frac{\mu_i}{\sum_{l=1}^{i} \mu_l} \frac{1}{\mu_i},$$

and we have the result by induction. For $k > R_i$,

$$v_k^i = \frac{1}{\sum_{l=1}^{i} \mu_l} + v_{k-1}^i,$$

and again the result follows by induction.

Corollary 2. *It holds that $T_j(\lambda) \leq T_j(0) = R_j$.*

2.2. Extensions

The advantage of our argument above as opposed to a standard dynamic programming argument is that it is easy to extend our results in many directions, such as impatient customers, discounted cost, batch arrivals and batch services, geometric services, general interarrival and service times, and random environments. Note that we still assume that there are secondary customers so that servers rejected by primary customers always become busy.

2.2.1. *Impatient customers.* An argument similar to that of Theorems 1 and 2 also shows that the threshold structure of the optimal policy is preserved for a system with impatient customers where the individual objective for a job is to maximize the probability of completing service before its deadline, and the social objective is to maximize the proportion of customers served before their deadlines. In particular, suppose that each customer will leave or renege after an exponentially distributed time (the deadline) with rate γ. Note that if we assume that the customers leave only while in the queue (i.e. they do not leave once they are at a server), then it is IO and SO for a customer to use any server that becomes available to minimize the waiting time in the queue, so the problem becomes trivial. However, if customers may also leave during service, there is a trade-off between waiting in the queue for a faster server and using an available but slower server. Let

$$T_j = \arg\min\left\{k\colon \frac{\mu_j}{\gamma + \mu_j} \geq p_k\right\},$$

where p_k is the probability of job k getting served before its deadline under the LCFP-IO policy when all servers are busy, and $\mu_j/(\gamma + \mu_j)$ is the corresponding probability if it accepts server j. To see that $T_j < \infty$, note that

$$p_k \leq \left[\prod_{i=1}^{k} \frac{(m+k-i)\gamma + m\mu_1}{(m+k+1-i)\gamma + m\mu_1}\right]\left[\frac{\mu_1}{\gamma + \mu_1}\right],$$

which would be the probability of job k getting served before its deadline under the IO policy if all servers had service rate equal to μ_1 and there were no new arrivals. This converges to 0 as k goes to ∞. Our prior argument shows that if there are at least T_j jobs in the queue, it is IO for job T_j to accept server j, and it is SO to assign some job to server j.

2.2.2. *Discounting.* If we are interested in minimizing the expected discounted waiting times, with no impatience, we can think of the discounted expected waiting time as the expected waiting time that occurs before a common random deadline that has an exponential distribution with rate equal to the discount rate, say γ. Again, let $T_j = \arg\min\{k\colon \mu_j/(\gamma + \mu_j) \leq v_k\}$, but now v_k is the waiting time before the common deadline for job k under the IO policy when all servers are busy. As before, it is both IO and SO to assign job T_j to server j. Similarly, we have the same threshold-type optimal policy when both impatience and discounting are included in the model.

2.2.3. *Batch arrivals.* Suppose that we have batch Poisson arrivals, so each arrival carries several jobs to be served, and the number of jobs in each arrival is independent and identically distributed with mean β. We assume that $\lambda\beta < \sum \mu_i$ for stability. When there is a new batch arrival, without loss of generality, the jobs in the batch are numbered relative to each other arbitrarily. As long as we otherwise index the jobs in descending order of their arrival times, the arguments in the proofs of Theorems 1 and 2 hold.

2.2.4. *Batch service.* Suppose that servers can process multiple jobs at the same time, i.e. server j can take as many as b_j jobs in a batch and finish them all after a time that is exponentially distributed with rate μ_j. Then, we can define the v_is as before (though their value will change to take into account the batch processing), and use them to determine the LCFP-IO policy. In particular, if server j is available, and if k is the lowest index such that $v_k \geq 1/\mu_j$, then jobs $k, \dots, k + b_J - 1$ will accept server j (if that many jobs are present). Also, the batch size for the lth time server j becomes available could be a random variable, say B_{jl}, as long as the B_{jl}s are independent for all $j = 1, \dots, m$ and $l = 1, 2, \dots$, and, for each j, the B_{jl}s are identically distributed for $l = 1, 2, \dots$.

2.2.5. *Geometric service times.* We can also extend the argument to a discrete-time model with geometric interarrival and service times. Note that now it is possible to have more than one available server because several servers can finish their jobs at the same time. However, the decision to use a server is independent of the states of the slower servers. To see this, suppose that server j_1 is the fastest available server. If job i is offered server j_1 and rejects it because $v_i \leq 1/\mu_{j_1}$ under the LCFP-IO policy, where v_i is defined as before, it will also reject all servers that are slower than server j_1, and will wait in the queue. Note that all servers become busy with a primary or secondary job before job i makes another decision, and all servers start a memoryless service each time, so the argument in Theorem 1 is still valid for this model. In particular, if k is the lowest index such that $v_k \geq 1/\mu_{j_1}$, job k will accept server j_1, and job $k + 1$ will become job k and will be offered the next fastest available server (say j_2). It will accept that server if and only if $v_k \geq 1/\mu_j$. The procedure continues until the lowest primary job decides or there are no more servers available. Any servers rejected by all primary jobs will be assigned secondary jobs, i.e. all servers become busy. Thus, if $j_1 < j_2 < \cdots < j_n$ are the available servers, and there are currently k customers in the queue, under the SO policy, a job will be assigned to server j_l if and only if $k - l + 1 \geq T_{j_l}$, where

$$T_j = \arg\min\left\{k : \frac{1}{\mu_j} \leq v_k\right\}.$$

We could also permit batch services in this geometric model and get similar results.

2.2.6. *General service and arrival distributions.* When the interarrival time and service times have general distributions (so arrivals form a renewal process), a threshold policy will still be both IO and SO. However, the expected waiting time for a job (and, therefore, the thresholds) will depend on the elapsed interarrival time and attained service times. In other words, the threshold functions will be state dependent. In particular, let m_i be the mean service time of server i, and let s be the current state of the system (the set of available servers, the attained service times for the unavailable servers, and the time since the last arrival), and suppose that server i is the fastest available server. If it is offered to job k under LCFP, job k will accept it as long as $m_i \leq v_k(s)$, where $v_k(s)$ is the expected total time in the system for job k in state s under LCFP, assuming that job k does not accept server i and server i becomes busy with another (possibly secondary) job. It is clear that since jobs $1, \dots, k - 1$ have priority over job k, $v_k(s)$ is increasing in k, so the LCFP-IO policy is a threshold policy. The argument that it is also SO is exactly the same as in the proof to Theorem 2, because deviating from the IO policy once and then following it thereafter makes all jobs worse off than following the IO policy from the beginning.

2.2.7. *Random environments.* We can also have a random environment, with a state X that changes according to an arbitrary Markov process (or, more generally, a time-dependent Markov process). The speeds of the servers may depend on X as well as the time to the next arrival. The value of accepting a server will be the total expected waiting and service times starting in the current state X, and appropriately taking into account possible future state changes before service completion. Again our argument will go through, with the LCFP-IO policy having *state-dependent* thresholds for accepting an available server, and those thresholds are indeed thresholds for the SO policy. We can argue as before that it will not be SO to force the lowest priority customer to accept a server it does not want, nor will it be SO to assign a secondary customer to a server that the lowest priority customer would want.

2.3. Computing the optimal thresholds

We now describe an iterative method to determine the optimal thresholds. We also use our results from this section in the next section. For simplicity, we consider $m = 2$ servers; the general analysis is similar though more complicated.

Let v_i^k be the waiting time for job i, assuming that jobs are ordered in the LCFS order, all servers are busy, and the kth job will accept server 2 if the next event is a completion on server 2 (i.e. when the threshold for using server 2 is k). Then, under the IO policy, $v_i = v_i^T$, where $T = T_2(\lambda) = \arg\min\{k\colon 1/\mu_2 \le v_k\}$.

From Corollary 2 we know that $1 \le T \le R_2 = T_2(0) = \lceil \mu_1/\mu_2 \rceil - 1$, so start with $t_0 = R_2$ as an initial guess for the optimal threshold. We then compute $v_k^{t_0}$, $k = 1, 2, \ldots$, as described below. If $t_0 = \arg\min\{k\colon 1/\mu_2 \le v_k^{t_0}\}$, we are done, otherwise we repeat the process with $t_1 = t_0 - 1$ as our current guess for the threshold. From Theorem 2 we will find some t_i between 1 and t_0 such that $t_i = \arg\min\{k\colon 1/\mu_2 \le v_k^{t_i}\}$.

We compute v_k^t for a threshold t as follows. First note that, using threshold t for $k > t$, the waiting time for job k is the total time until the job becomes job t plus v_t^t. While job k is waiting to ‘move up’ to position t, both servers will be busy and the waiting time, under LCFP, is just the sum of $k - t$ busy periods for an M/M/1 queue with arrival rate λ and service rate $\mu_1 + \mu_2$. That is, for $k > t$,

$$v_k^t = \frac{k-t}{\mu_1 + \mu_2 - \lambda} + v_t^t. \tag{2}$$

For $1 < k < t$, by conditioning on the first event we have

$$v_k^t = \frac{1}{\mu_1 + \lambda}(1 + \lambda v_{k+1}^t + \mu_1 v_{k-1}^t),$$

and, for $k = 1 < t$,

$$v_1^t = \frac{1}{\mu_1 + \lambda}\left(1 + \lambda v_2^t + \mu_1 \frac{1}{\mu_1}\right) = \frac{1}{\mu_1 + \lambda}(2 + \lambda v_2^t).$$

Similarly, for $k = t > 1$,

$$\begin{aligned} v_t^t &= \frac{1}{\mu_1 + \mu_2 + \lambda}\left(1 + \lambda v_{t+1}^t + \mu_1 v_{t-1}^t + \mu_2 \frac{1}{\mu_2}\right) \\ &= \frac{1}{\mu_1 + \mu_2 + \lambda}\left(2 + \lambda\left[\frac{1}{\mu_1 + \mu_2 - \lambda} + v_t^t\right] + \mu_1 v_{t-1}^t\right), \end{aligned}$$

so

$$v_t^t = \frac{1}{\mu_1 + \mu_2}\left(\frac{2(\mu_1 + \mu_2) - \lambda}{\mu_1 + \mu_2 - \lambda} + \mu_1 v_{t-1}^t\right).$$

Finally, for $k = t = 1$,

$$v_1^1 = \frac{1}{\mu_1 + \mu_2 + \lambda}\left(1 + \lambda\left[\frac{1}{\mu_1 + \mu_2 - \lambda} + v_1^1\right] + \mu_1 \frac{1}{\mu_1} + \mu_2 \frac{1}{\mu_2}\right),$$

so

$$v_1^1 = \frac{1}{\mu_1 + \mu_2}\left(\frac{3(\mu_1 + \mu_2) - 2\lambda}{\mu_1 + \mu_2 - \lambda}\right). \tag{3}$$

The equations above can be solved recursively.

2.4. The effect of server heterogeneity

We now consider the effect of changing the server speeds when there are two servers, subject to holding the total service rate constant. To build intuition, let us first consider the model with no secondary jobs. Singh and Prasad [14] considered the standard M/M/2/N queue with $\mu_1 > \mu_2$, and both λ and $\rho = \lambda/(\mu_1 + \mu_2) < 1$ held fixed. Here, jobs always go to server 2 when it is available and server 1 is not; they go to server 1 if both are available. They showed that, for all N, the optimal value of μ_2 to stochastically minimize the steady-state number in the system is

$$\mu_2^* = \lambda\left(\left(1 + \frac{1}{\rho}\right)^{1/2} - 1\right).$$

One might think that $\mu_2^* = 0$ would be optimal, because the number in the system is stochastically smaller in an M/M/1/N queue with rate $\mu_1 + \mu_2$ than for the M/M/2/N queue with $\mu_1 > \mu_2$ for any N and μ_2, but, in this two-server system, it is assumed that jobs always go to server 2 when it is available, regardless of its speed. Thus, as μ_2 approaches 0, the delay for a few jobs (and, hence, the mean delay) goes to ∞. On the other hand, if we use the optimal threshold for sending jobs to server 2 in the model with two servers and no secondary jobs, it is clear that $\mu_2^* = 0$, the slow server is never used, and we have a single fast server at maximal rate.

We now show that in the model with secondary jobs, $\mu_2^* = 0$ if we use the optimal threshold. We also partially characterize how the mean waiting time changes with respect to μ_2 when the total service rate is fixed to a constant (i.e. $\mu_1 + \mu_2 = M$). Based on our results, we present a conjecture that the mean waiting time is increasing in μ_2, i.e. better balanced service rates are worse in terms of the mean waiting time.

From now on we assume that $M = \mu_1 + \mu_2$ is held fixed, and that $0 \le \mu_2 \le \mu_1 \le M$ (so $\mu_2 \le M/2$). Let $W(\mu_2)$ be the mean waiting time for a random arrival under the SO policy, and let $v_k(\mu_2)$ be the expected waiting time of the kth customer under the LCFP-IO policy. From Theorem 2, $W(\mu_2) = v_1(\mu_2)$, which is the mean waiting time for a job from the time it arrives under the LCFP-IO policy, i.e. under the SO policy.

Proposition 1. *It holds that* $W(0) = (2M - \lambda)/(M(M - \lambda)) \le W(\mu_2)$. *That is, the mean waiting time is minimized when there is a single fast server at rate* $\mu_1 = M$.

Proof. When $\mu_2 = 0$, the departure process of primary jobs under the optimal (infinite) threshold is a Poisson process with rate M except when there is a secondary job on the single fast server. This occurs only when the system is empty of primary jobs, or after the first primary job arrival to such an empty system, before the service completes. Thus, a busy period looks like a standard M/M/1 busy period with exceptional first service, where the total service time of the first arrival to the busy period is the sum of two exponential random variables with rate M. (Here, without loss of generality for the mean waiting time, we assume an FCFS service discipline.) When $\mu_2 > 0$, the first customer to a busy period must still wait for a server to become available, which is exponential at rate M. Thereafter, during the busy period the rate of departures is never more than M, and will be strictly less than M whenever there is only one primary job at a server. Thus, we can construct coupled sample paths for the systems with $\mu_2 = 0$ and $\mu_2 > 0$, so that all departures are stochastically earlier for the $\mu_2 = 0$ system.

Now consider $W(0) = v_1(0)$, the mean waiting time for a single-server queue with rate M. Here $v_k(0)$ is the sum of $k-1$ busy periods with service rate M, plus $v_1(0)$, and $v_1(0)$ satisfies

$$v_1(0) = \frac{1}{M+\lambda}\left(1 + \lambda v_2(0) + M\frac{1}{M}\right) = \frac{1}{M+\lambda}\left(2 + \lambda\left[\frac{1}{M-\lambda} + v_1(0)\right]\right)$$

so

$$v_1(0) = \frac{2M-\lambda}{M(M-\lambda)} = W(0).$$

We have the following conjecture that waiting times increase when server speeds are better balanced, and we give some evidence for it below.

Conjecture 1. *It holds that $W(\mu_2)$ is increasing in μ_2, $0 \le \mu_2 \le M/2$.*

Let $v_k^t(\mu_2)$ be the expected waiting time of the kth customer under the LCFP policy when the threshold for using the slow server is t and the service rate of server 2 is μ_2. Suppose that we use a threshold of 1, i.e. we always use both servers. Then, from the analysis in the previous section (see (2) and (3)), we have

$$v_1^1(\mu_2) = \frac{1}{\mu_1+\mu_2}\,\frac{3(\mu_1+\mu_2)-2\lambda}{\mu_1+\mu_2-\lambda} = \frac{1}{M}\,\frac{3M-2\lambda}{M-\lambda}$$

and

$$v_k^1(\mu_2) = \frac{k-1}{M-\lambda} + \frac{1}{M}\,\frac{3M-2\lambda}{M-\lambda} = \frac{(k+2)M-2\lambda}{M(M-\lambda)}.$$

Also, the waiting time for a random arrival using a threshold of 1 is $v_1^1(\mu_2)$. Note that when a threshold of 1 is used, whether optimal or not, the performance depends only on the sum $M = \mu_1 + \mu_2$ and not on the individual values of μ_1 and μ_2 for $0 < \mu_2 \le \mu_1$, i.e. $v_1^1(\mu_2) \equiv v_1^1 := (3M-2\lambda)/M(M-\lambda)$. This is in contrast to the model without secondary jobs with a threshold of 1 [14]. With secondary jobs, we can never have arrivals who find a server idle.

The optimal threshold is 1 if $1/\mu_2 \le v_1^1$, which is equivalent to

$$\mu_2 \ge \frac{M-\lambda}{2M-\lambda}\mu_1 = \frac{1-\rho}{2-\rho}\mu_1.$$

This follows, with a little algebra, because $\mu_2 \ge 1/v_1^1 = M(M-\lambda)/(3M-2\lambda) = (\mu_1 + \mu_2)(M-\lambda)/(3M-2\lambda)$. Thus, both servers will always be used if the fast server is less than twice as fast as the slower server, and if the system is heavily loaded, the slow server must be very slow for it to be optimal to not always use it. For example, if $\rho = 0.9$, the faster server must be at least eleven times faster for it to be better to not use the slow server when a primary job is waiting.

If $\mu_2 < 1/v_1^1 = M(M-\lambda)/(3M-2\lambda)$ (or, equivalently, $\mu_2 < (1-\rho)\mu_1/(2-\rho)$) then 1 is not the optimal threshold, and

$$W(\mu_2) \le v_1^1 = \frac{1}{M}\,\frac{3M-2\lambda}{M-\lambda} = W(0) + \frac{1}{M}.$$

Suppose that we use a threshold of 2. From the analysis in the previous section,

$$v_1^2(\mu_2) = \frac{2M^2 - \lambda^2}{(M-\lambda)[M\lambda + (M-\lambda)\mu_1]},$$

which is decreasing in μ_1 and, therefore, increasing in μ_2, holding $\mu_1 + \mu_2 = M$ constant. Similarly, for a threshold of 3, we obtain

$$v_1^3(\mu_2) = \frac{3M^2\lambda - \lambda^3 - M\lambda^2 - 4M\lambda\mu_1 + 2M^2\mu_1 + 2\lambda^2\mu_1}{\lambda^3\mu_1 - M\lambda^3 - 2M\lambda\mu_1^2 - 2M\lambda^2\mu_1 + M^2\lambda\mu_1 + M^2\lambda^2 + M^2\mu_1^2 + \lambda^2\mu_1^2}.$$

Taking the derivative with respect to μ_1, we obtain something proportional to

$$\begin{aligned} g(\mu_1) := &-2(M-\lambda)^2\mu_1^2 - 2\lambda(3M^2 - M\lambda - \lambda^2)\mu_1 \\ &+ 2(M-\lambda)M\lambda^2 - \lambda^2(3M^2 - M\lambda - \lambda^2), \end{aligned}$$

so $v_1^3(\mu_2)$ will be decreasing in μ_1 (increasing in μ_2) if we can show that $g(\mu_1) \le 0$. First note that $g'(\mu_1)$ is negative, and that $\mu_1 > \lambda/2$ because we assume that $M > \lambda$ for stability, and $\mu_1 \ge \mu_2$. Therefore, it is sufficient to show that $g(\lambda/2) \le 0$, which is easily checked, using the fact that $M > \lambda$. Thus, whenever μ_2 is such that the optimal threshold is 1, 2, or 3, $W(\mu_2) = v_1^1(\mu_2)$, $W(\mu_2) = v_1^2(\mu_2)$, or $W(\mu_2) = v_1^3(\mu_2)$, respectively, and, therefore, is increasing in μ_2. We can obtain a closed-form expression for $v_1^t(\mu_2)$ from the system of equations in the previous section for general t, but the expression is quite messy and it is not easy to see the monotonicity of $v_1^t(\mu_2)$ in the general case.

Now let us consider the case as $\lambda \to 0$. In this case, the probability that a random arrival (which has highest priority under LCFP-IO) will lose its position before being served goes to 0, i.e. $W(\mu_2) \to \min\{3/M; 2/\mu_1\}$ (depending on whether the threshold is 1 or not), which again is decreasing in μ_1 and increasing in μ_2.

Summarizing, we have the following partial support for our conjecture.

Proposition 2. *For* $M(M-\lambda)/(3M-2\lambda) \le \mu_2 \le M/2$, $W(\mu_2)$ *is constant in* μ_2 *and the optimal threshold is* 1. *Whenever the optimal threshold is* 1, 2, *or* 3, *and in the limit as* $\lambda \to 0$, $W(\mu_2)$ *is increasing in* μ_2. *Also, for* $0 \le \mu_2 \le M/2$,

$$\frac{2M-\lambda}{M(M-\lambda)} = W(0) \le W(\mu_2) \le W\left(\frac{M}{2}\right) = \frac{1}{M}\frac{3M-2\lambda}{M-\lambda} = W(0) + \frac{1}{M}.$$

3. The model with primary jobs only

Including an infinite supply of secondary jobs in the model of the last section simplified the argument, and permitted us to prove that the optimal policy is a threshold type. Using an intuitive argument, we first showed that the IO policy is of a threshold type, then proved that the IO policy was in fact an SO policy. Moreover, there was a single threshold for each server, because with secondary jobs there is never more than one available server. In this section we explore whether the same machinery can be extended to the case with no secondary jobs, i.e. the original job assignment problem with multiple heterogeneous servers. First, we consider the IO policy under Xu's LCFP-P policy, in which jobs that arrive later have higher priority, and higher priority jobs can preempt lower priority jobs at servers. We show that the IO policy is still a threshold policy, where the thresholds depend on the relative indices of jobs at the servers. We also show that with more than two servers, the IO policy under LCFP-P is no longer SO.

We then extend Walrand's argument to show the existence of a SO threshold function that determines when to use server j, when $\mu_1 = \mu_2 = \cdots = \mu_{j-1} > \mu_j$. We show why the arguments based on the work of Walrand [17] and Xu [18] do not apply in general when there are more than two servers in the system.

3.1. The IO policy under LCFP-P

We recall the definition of LCFP-P from Xu [18].

Definition 1. (*Xu [18].*) A service discipline is called LCFP-P if it grants priorities to jobs as follows.

(i) At any time all jobs in the system are numbered in descending order of their arrival times. A job with a lower index possesses a higher priority to access servers.

(ii) A lower indexed (higher priority) job, while waiting in the queue, can preempt the service of a higher indexed (lower priority) job. A job cannot abandon its server unless it is preempted by a lower indexed job.

Let $\alpha = (\alpha_1, \dots, \alpha_m)$ be the state of the servers, where $\alpha_j \in \mathbb{Z}^+$, $j = 1, \dots, m$, is the index of the job at server j, and $\alpha_j = \infty$ if server j is idle. In contrast to the model with secondary jobs, here we must keep track of the indices for all jobs in the system, not just those in the queue. Let $V_i(\alpha)$ be the expected remaining time in the system of job i in server state α under the IO policy. Note that, under LCFP-P, a job needs to consider only the jobs that arrived or will arrive later than itself when deciding its IO policy. Furthermore, the decision of a job is indifferent to which lower indexed jobs are in service or in the queue, i.e. it depends only on the number of those jobs.

Lemma 1. *We have $V_i(\alpha) = V_i(\hat{\alpha})$ for any i, and any α, $\hat{\alpha}$ such that $\hat{\alpha}_j < i$ if $\alpha_j < i$ and $\hat{\alpha}_j > i$ if $\alpha_j > i$. (In the latter case, we may have α_j or $\hat{\alpha}_j$ equal to ∞, i.e. server j is idle.)*

We can partially characterize the IO policy under LCFP-P, though we have not been able to show that the IO policy is defined by thresholds for all states α.

Theorem 3. *The IO policy has the following properties.*

(i) *The lowest indexed job will always use server 1. In particular, a new arrival will always use server 1.*

(ii) *When the IO policy assigns a job to a server, it always uses the fastest available server.*

(iii) *If the queue length becomes large enough, all servers will be used.*

Weber [16] and Rosberg and Makowski [12] showed that Theorem 3(i) and (ii) hold for the SO policy.

To prove Theorem 3, we need the following lemma. Let F and S indicate two servers, the faster server and the slower server, respectively, i.e. $\mu_F > \mu_S$, $F, S \in \{1, \dots, m\}$ and $F < S$. Let α_F and α_S represent states when job i, the tagged job, is at server F and a higher indexed job k $(> i)$ is at server S, and the other way around, respectively. Let the states of the other servers be arbitrary but the same for both α_F and α_S.

Lemma 2. *If all jobs follow the IO policy then being on the faster server always gives the tagged job smaller expected cost than being on the slower server. That is, $V_i(\alpha_F) \leq V_i(\alpha_S)$.*

Proof. Let $\mathbb{S}_F$ and $\mathbb{S}_S$ respectively represent the system starting in state α_F and α_S with $x \geq i$ jobs in the system. For convenience, we call job i in both systems the *tagged job*. We consider the discrete-time uniformized model, with a uniformization constant (mean time in state) of 1, i.e. we scale time so that $\lambda + \sum \mu_j = 1$. Our proof is by induction on a sequence of finite-horizon problems. Now the problem is equivalent to a discrete-time dynamic programming problem. Let $V_i^n(\alpha)$ be the expected remaining time in the system of the tagged job in state α under the IO policy when there are n transitions left. Suppose that $V_i^n(\alpha_S) \geq V_i^n(\alpha_F)$ for all i, and α_S and α_F. For $n = 1$, this holds trivially because $V_i^1(\alpha_S) = V_i^1(\alpha_F) = 1$ for all i. We now show that it holds for $n+1$. Note that a consequence of the induction hypothesis is that Theorem 3(ii) holds for n. Let A and C_j represent the possible transitions: an arrival and a job completion at server j. We couple the events in the two systems as follows. With probability μ_S, a job completion occurs at server F, C_F, in $\mathbb{S}_F$ and a job completion occurs at server S, C_S, in $\mathbb{S}_S$ (case (i)). With probability μ_S, C_S occurs in $\mathbb{S}_F$ and C_F occurs in $\mathbb{S}_S$ (case (ii)). With probability $\mu_F - \mu_S$, C_F occurs in both systems (case (iii)). With probability λ, A occurs in both systems, and, with probability μ_i, C_i, $i \notin \{F, S\}$, occurs in both systems. If the first transition is case (i), the tagged job leaves both systems, so that $V_i^{n+1}(\alpha_S) = V_i^{n+1}(\alpha_F) = 1$. If it is case (iii), the tagged job leaves the system in $\mathbb{S}_F$ but not in $\mathbb{S}_S$, i.e. $V_i^{n+1}(\alpha_S) > V_i^{n+1}(\alpha_F) = 1$. For all other events, including case (ii), the tagged job is still in the system for both systems. If jobs with lower index than the tagged job use an available server after the first transition in $\mathbb{S}_F$, this also happens in $\mathbb{S}_S$. Such jobs consider both servers F and S available. From the induction hypothesis applied to the lower indexed job and Lemma 1, either (a) both servers S and F will be used, (b) only server F will be used, or (c) neither F nor S will be used. In case (a) job i will be back in the queue and the states will be the same, so $V_i^{n+1}(\alpha_S) = V_i^{n+1}(\alpha_F)$. In case (b), since the tagged job in $\mathbb{S}_F$ can choose server S or not, while in $\mathbb{S}_S$ it has to stay at server S, we have $V_i^n(\alpha_S) \geq V_i^n(\alpha_F)$. In case (c), the states will be the same except that the tagged job is at server F and a higher indexed job k ($k = \infty$ in case (ii)) is at server S in $\mathbb{S}_F$, and vice versa in $\mathbb{S}_S$. Hence, in this case, from the induction hypothesis, $V_i^{n+1}(\alpha_S) \geq V_i^{n+1}(\alpha_F)$. Therefore, in all cases, $V_i^{n+1}(\alpha_S) \geq V_i^{n+1}(\alpha_F)$, and letting $n \to \infty$ (with $\lambda < \sum \mu_j$), $V_i(\alpha_S) \geq V_i(\alpha_F)$.

Proof of Theorem 3. (i) From Lemma 2 we know that if job 1 uses a server, it will use server 1. A simple induction argument shows that using server 1 immediately is better than staying in the queue for job 1.

(ii) This is a direct corollary of Lemma 2.

(iii) We suppose that, under the IO policy, if a job is offered server j and it is indifferent between using it and not using it, then it uses it. Then $V_i(\alpha)$ is strictly increasing in i for all α, since job i will only be offered servers that are strictly suboptimal for job $i-1$. By way of contradiction, let us suppose that server j is never used under the IO policy. Since $V_i(\alpha)$ is strictly increasing in i, there will be some i such that $V_i(\alpha) > 1/\mu_j$, where α is such that $\alpha_j = \infty$, so that customer i will be better off accepting server j (and staying on until completion since we suppose that it is not optimal for other customers to use j). But this is in contradiction with the optimality of a policy that never uses j.

3.2. The limitations of Xu's approach for the general problem

Now we show that Xu's argument [18] for the equivalence of the IO and SO policies does not work with three or more servers. The main idea in the proof by Xu for two servers is as follows. Let the last job in the queue be the tagged job. Since this is the lowest priority job, if a policy does not agree with the IO policy in terms of assigning a job to server 2 for the first decision, but agrees with IO thereafter, we can do better by applying the IO policy for the tagged job for the first decision, because its decision will have no effect on the other jobs. When there are three (or more) servers, the last job in the queue (call it the tagged job) may not be the lowest priority job; there could be a job with lower priority on server 3 when server 2 is available to the tagged job. Assigning the tagged job to server 2 can then affect the waiting time of the lower priority job on server 3.

Theorem 4. *The IO policy is not necessarily the SO policy.*

Proof. Suppose that server j is the fastest available server. Note that, under the SO policy, all jobs are indistinguishable, so that its decision for assigning a job to server j depends only on the queue length and which of the slower servers are idle. Therefore, for the IO policy to be SO, the IO policy for job i in the queue must make the same decision regardless of which jobs are assigned to slower servers, i.e. it must make the same decision for job i when job k is assigned to a slower server, regardless of whether $k < i$ or $k > i$. Since the decision in the latter case is the same as when the slower server is idle ($k = \infty$), for the IO policy to be optimal, it must make the same decision regardless of whether slower servers are idle or not. This means that if the IO and SO policies are the same, then the SO policy does not depend on the states of the slower servers. However, Weber [16] found an example in which the policy for assigning a single job in the queue to the second fastest server in a three-server model depends on whether the third server is idle or not. Hence, for this case, the SO policy cannot be the same as the IO policy.

3.3. Partial structure of the SO policy

Let $(k, \bar{\alpha}) \in \mathbb{Z} \times \{0, 1\}^m$ represent the state in which there are k jobs in the queue, where $\bar{\alpha} = (\bar{\alpha}_1, \dots, \bar{\alpha}_m)$ is a vector of dimension m, and $\bar{\alpha}_j$ is 0 or 1 as server j is idle or busy, respectively. Because we are now only considering the SO policy, we need not keep track of the indices of jobs on servers because the SO policy is independent of the job identities (indices).

Suppose that the fastest f servers all have the same service rate. Rosberg and Makowski [12] and Weber [16] showed the following result for a single fast server. It is easily extended to multiple fast servers with the same rate.

Lemma 3. (i) *The fastest f servers should always be used if possible.*

(ii) *When a job is assigned to a server, it is always assigned to the fastest available server.*

We also consider the following special case and show that a state-dependent threshold policy minimizes the expected waiting time. Suppose that the fastest f servers all have the same service rate, and that they are busy and all of the other $m - k$ servers are idle, i.e. the server availability state is $(1, 1, \dots, 1, 0, 0, \dots, 0)$, where the number of 1s (busy servers) is f. Let us call this state $\bar{\alpha}_f$. It can be shown that the optimal policy to assign a job to server $f + 1$ is a threshold type. The proof uses the idea of Walrand's proof for two servers [17].

Theorem 5. *When the server availability state is $\bar{\alpha}_f$, there exists a threshold $T_{f+1}(\bar{\alpha}_f)$ such that it is optimal to use server $f + 1$ if and only if $k \geq T_{f+1}(\bar{\alpha}_f)$.*

Proof. First note that there exist states such that server $f + 1$ is used, because otherwise, for a long enough queue, the last job would be better off using $f + 1$ even if no one else ever used it, leading to a contradiction. Now suppose, by way of contradiction to the threshold structure, that the optimal policy π uses server $f + 1$ in states $(x, \bar{\alpha}_f)$ and $(z, \bar{\alpha}_f)$ for two noncontiguous numbers, i.e. x and z such that $x + 1 < z$, but π does not use server $f + 1$ in state $(y, \bar{\alpha}_f)$ for some y, $x < y < z$. Let the initial state be $(y, \bar{\alpha}_f)$. That is, π will only use the fastest servers $1, \dots, f$ until the queue length leaves $\{x+1, \dots, z-1\}$ by Lemma 3(ii) and it will use all f fastest servers by Lemma 3(i). Pick an arbitrary job in the initial state and call it the tagged job. Give it the lowest priority until the queue length leaves $\{x+1, \dots, z-1\}$. Note that this has no effect on the mean waiting time from the system's perspective. When the queue length leaves $\{x+1, \dots, z-1\}$, π will use server $f+1$, and assume without loss of generality that the job that will first use server $f+1$ is the tagged job. Then, we would strictly reduce the mean waiting time for the tagged job without affecting the waiting times for any other job by sending the tagged job to server $f+1$ at time $t = 0$ instead of waiting for some positive random time before taking the same action anyway.

3.4. The limitations of Walrand's approach for the general problem

Walrand's key idea is that if a threshold policy is not optimal, i.e. if there were a gap in the queue length state where server 2 (or server $f+1$ in our extension above) is not used, but it is used for shorter and longer queue lengths, then, with probability 1, the state will leave such a gap before the queue empties, and will at that point use the slower server. It follows that it would be better to use server 2 from the beginning. However, when there are more than two servers in the system, it is possible to define gaps such that the slow server is not used in state $(y, \bar{\alpha})$, and it is used in states $(x, \bar{\alpha})$ and $(z, \bar{\alpha})$, $x < y < z$, and such that it is possible for the queue length to leave the gap and go to 0, without going through either states $(x, \bar{\alpha})$ or $(z, \bar{\alpha})$, and without using the slower server. For example, suppose that there are three servers in the system, and suppose that a proposed optimal policy is such that, when $\bar{\alpha} = (1, 0, 1)$, we do use server 2 when there are x jobs in the queue, and also when there are at least z jobs in the queue; however, we do not use server 2 in between, i.e. for $x+1, \dots, z-1$ jobs in queue. Also, suppose that when $\bar{\alpha} = (1, 0, 0)$, we do not use server 2 if there are fewer than z jobs in the queue. Then, if the initial state is $(y, 1, 0, 1)$, where $x < y < z$, and the first transition is a job completion on server 3, the queue length leaves the gap, and yet we will not use server 2 at this point because the server state has changed to $(1, 0, 0)$. Now, it is not possible to construct a better policy that uses the slow server immediately than the policy that waits until the queue length leaves the gap. We conjecture that the optimal policy will not have a gap structure as suggested above, but we cannot disprove its optimality with Walrand's approach.

3.5. Extensions for the two-server model

When there are only two servers in the system, a sample path argument can show that the threshold-type optimal policy is preserved for most of the extensions we considered in Section 2.2 for the multiserver two-class system. In particular, we can permit impatient customers, batch arrivals, discounted cost, and geometric service times. For example, in a model with impatient customers, where each customer or job will leave after an exponentially distributed time with rate γ, we still have a birth–death process for the queue length while server 2 is not being used. Therefore, a sample path argument like Walrand's shows that the optimal policy is a threshold type because any gap in the queue length state for using server 2 would not be optimal.

We cannot extend the result to batch services, if we are permitted to start service with a partial batch, and we assume no more jobs can be assigned to the server before the partial batch completes. Walrand's argument will not work because a batch service can take us out of a gap in which server 2 is not used (the Markov chain is no longer skip-free to the left), to a small state in which it is not optimal to use server 2, and from which the system may empty before moving to a state in which it is optimal to use server 2. Xu's approach will also not work, because, under the LCFS-P policy, the lowest priority job may be the only one that wants to use a server, but if it uses it, it makes the other positions in the batch unavailable for future arrivals. So, for example, if there is a single job in the queue and server 1 is available and has batch size 2, then it is not hard to show that the IO policy for the single job is to use the server. If we have two arrivals before server 1 completes, the second arrival cannot use server 1, so is harmed by the original job's initial decision. This violates a key assumption in Xu's argument, that making the IO decision for the lowest priority job has no effect on other jobs.

4. Conclusion

We have shown that the IO policy is a threshold-type policy and that it is also the SO policy in a model with an infinite supply of secondary jobs, and where the objective is to minimize the mean waiting time for primary jobs. We also provided an algorithm to compute the optimal policy. For the classical heterogeneous server problem, without secondary jobs, we gave extensions for the two-server case, and strengthened the known [12], [16] partial structural results when there are more than two servers. However, none of the existing arguments for the two-server case work with more than two servers to show the complete result, i.e. the optimal policy is a state-dependent threshold policy. It also does not seem possible to extend the dynamic programming approach to the m-server system for $m \geq 3$. Thus, providing an intuitive probabilistic proof for the complete structure of the optimal policy for the original job assignment problem still remains to be done.

References

[1] Agrawala, A. K., Coffman, E. G., Jr., Garey, M. R. and Tripathi, S. K. (1984). A stochastic optimization algorithm minimizing expected flow times on uniform processors. *IEEE Trans. Comput.* **33,** 351–356.

[2] Armony, M. and Mandelbaum, A. (2011). Routing and staffing in large-scale service systems: the case of homogeneous impatient customers and heterogeneous servers. *Operat. Res.* **59,** 50–65.

[3] Armony, M. and Ward, A. R. (2010). Fair dynamic routing in large-scale heterogeneous-server systems. *Operat. Res.* **58,** 624–637.

[4] De Vericourt, F. and Zhou, Y.-P. (2006). On the incomplete results for the heterogeneous server problem. *Queueing Systems* **52,** 189–191.

[5] Koole, G. (1995). A simple proof of the optimality of a threshold policy in a two-server queueing system. *Systems Control Lett.* **26,** 301–303.

[6] Kumar, P. R. and Walrand, J. (1985). Individually optimal routing in parallel systems. *J. Appl. Prob.* **22,** 989–995.

[7] Lin, W. and Kumar, P. R. (1984). Optimal control of a queueing system with two heterogeneous servers. *IEEE Trans. Automatic Control* **29,** 696–703.

[8] Luh, H. P. and Viniotis, I. (2002). Threshold control policies for heterogeneous server systems. *Math. Meth. Operat. Res.* **55,** 121–142.

[9] Righter, R. (1988). Job scheduling to minimize expected weighted flowtime on uniform processors. *Systems Control Lett.* **10,** 211–216.

[10] Righter, R. and Xu, S. (1991). Scheduling jobs on heterogeneous processors. *Ann. Operat. Res.* **29,** 587–601.

[11] Righter, R. and Xu, S. H. (1991). Scheduling jobs on nonidentical IFR processors to minimize general cost functions. *Adv. Appl. Prob.* **23,** 909–924.

[12] Rosberg, Z. and Makowski, A. M. (1990). Optimal routing to parallel heterogeneous servers—small arrival rates. *IEEE Trans. Automatic Control* **35,** 789–796.

[13] Rykov, V. V. (2001). Monotone control of queueing systems with heterogeneous servers. *Queueing Systems* **37,** 391–403.

[14] Singh, V. P. and Prasad, J. (1976). A heterogeneous system with finite waiting space. *J. Eng. Math.* **10,** 125–134.

[15] Stockbridge, R. H. (1991). A martingale approach to the slow server problem. *J. Appl. Prob.* **28,** 480–486.

[16] Weber, R. (1993). On a conjecture about assigning jobs to processors of differing speeds. *IEEE Trans. Automatic Control* **38,** 166–170.

[17] WALRAND, J. (1984). A note on: "Optimal control of a queuing system with two heterogeneous servers". *Systems Control Lett.* **4,** 131–134.

[18] XU, S. H. (1994). A duality approach to admission and scheduling controls of queues. *Queueing Systems* **18,** 273–300.

[19] XU, S. H. AND SHANTHIKUMAR, J. G. (1993). Optimal expulsion control—a dual approach to admission control of an ordered-entry system. *Operat. Res.* **41,** 1137–1152.