12-1-2011

Dynamic Server Allocation at Parallel Queues

Susan E. Martonosi
Harvey Mudd College

Recommended Citation

Martonosi, Susan E. (2011, December). Dynamic Server Allocation at Parallel Queues. IIE Transactions, 43(12), 863-877.

Dynamic Server Allocation at Parallel Queues

Susan E. Martonosi
Harvey Mudd College
301 Platt Blvd.
Claremont, CA 91711
(909) 607-0481
martonosi@math.hmc.edu

Abstract: We explore whether dynamically reassigning servers to parallel queues in response to queue imbalances can reduce average waiting time in those queues. We use approximate dynamic programming methods to determine when servers should be switched, and we compare the performance of such dynamic allocations to that of a pre-scheduled deterministic allocation. Testing our method on both synthetic data and data from airport security checkpoints at Boston Logan International Airport, we find that in situations where the uncertainty in customer arrival rates is significant, dynamically reallocating servers can substantially reduce waiting time. Moreover, we find that intuitive switching strategies that are optimal for queues with homogeneous entry rates are not optimal in this setting. Keywords: control of queues, fluid queues, approximate dynamic programming, dynamic server allocation, workforce management.

1 Introduction

A common trade-off experienced in real-world queuing systems is balancing the need to keep waiting times low while minimizing the staffing required to serve the queues. We examine techniques for the efficient dynamic allocation of a pool of servers to spatially dispersed queues. Specifically, we examine a two-queue system with a nonhomogeneous arrival process, having a collection of servers, each of whom can be assigned to exactly one of the queues at any given time. As time progresses, the queue lengths might become unbalanced between the two queues such that one or more servers from one queue should be switched to the other. Because of the spatial distance between the two queues, switching one server from Queue A to Queue B incurs a time lag during which that server is unavailable at either queue, temporarily reducing the capacity of the queue.

We use approximate dynamic programming (ADP) techniques to find near-optimal anticipatory switching policies as a function of the system state, future arrival rates and switching times. We find that the benefit of dynamically switching servers between queues in response to stochastic fluctuations in queue traffic depends on uncertainty in the customer arrival rates. If there is much uncertainty in the passenger arrival rates, then the ability to flexibly respond to fluctuations in queue lengths by dynamically switching servers between queues can help reduce queue waiting times. We also show that the optimal switching policy is not well approximated by simple heuristics.

The queuing problem studied here is motivated by techniques observed at San Francisco International Airport (SFO) in 2004, where video cameras were focused on the different security checkpoints and projected their images onto screens in a central control center. Transportation Security Administration (TSA) officials examined the monitors, looking primarily for security breaches. However, if they found the queue at one of the checkpoints getting too long or suspected that it would grow quickly in the near future, they could decide to close a lane at a less busy checkpoint in the airport and transfer those screeners to a previously unstaffed lane at a busier checkpoint. This decision of when to switch was made based on current queue lengths, knowledge of future arrival rates to the checkpoint according to scheduled flight departures, and employee experience.

Although motivated by an airport security context, this queue control problem is useful in a variety of other applications as well. These include parallel border crossings, such as in the Niagara

Falls area of New York State, and department store checkout stations, which are often spatially distributed throughout the store.

In the following section, we describe the queuing decision problem and introduce parameters and terminology that are used throughout the paper. Section 3 relates our problem to work already conducted on similar queue control models. In Section 4, we present three different formulations for this problem: one in which both arrivals to the queue and service times are deterministic, one in which the arrival pattern is altered in a random fashion, and one in which service times are random. Section 5 addresses computational issues associated with these formulations and presents approximate dynamic programming techniques that are used in the analysis. In Section 6, we evaluate our models using both synthetic data and airport security checkpoint data from Massport's Boston Logan International Airport as a realistic case study. We discuss opportunities for further research and offer a few concluding remarks in Section 7.

2 Problem description

In this section, we introduce our key modeling framework, parameters and assumptions. We consider two queues, A and B, serving different sets of customers. Arrival processes to these queues are independent with time-varying, piecewise constant rates, $\lambda_A(t)$ and $\lambda_B(t)$. We assume that customers must wait in a particular queue, and therefore queue load balancing can be achieved only by switching servers, rather than passengers, between queues. This is realistic in the cases of airport security checkpoints and border crossings. At many airports, passengers must wait in the security checkpoint queue corresponding to their terminal and/or departure gate and cannot choose to wait in a shorter queue. Similarly, drivers arrive at a particular border crossing depending on their route, and are unlikely to have access to information that would permit them to change to a crossing point with a shorter wait.

A fixed number, N, of servers, can be allocated to the two queues, subject to the constraint that the number of servers at a given queue cannot exceed the number of service stations at that queue (e.g. checkout lanes at a department store, or the number of x-ray machines at an airport security checkpoint), which we denote N_{max_A} and N_{max_B}. Each server processes customers at rate

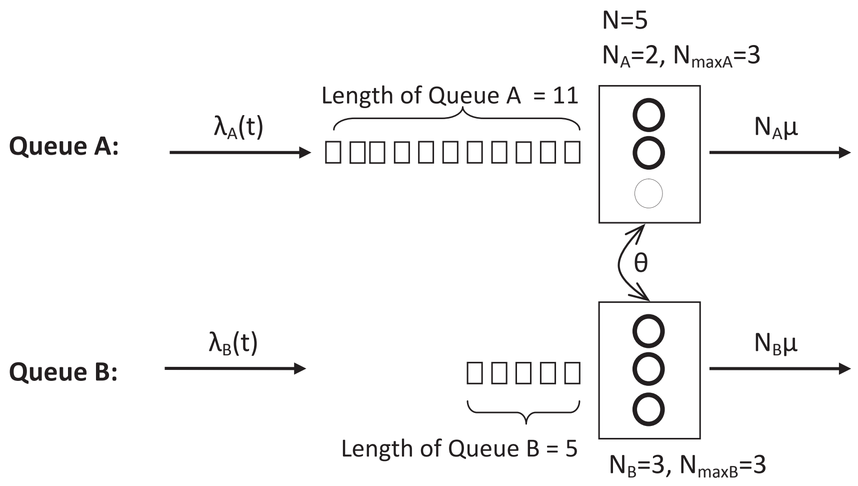

Figure 1: Schematic of the queuing system. There are two queues, A, and B, each having nonhomogeneous arrivals with rates $\lambda_A(t)$ and $\lambda_B(t)$. There is a pool of $N = 5$ servers that can be allocated to the two queues, and each queue can accommodate at most $N_{max_A} = N_{max_B} = 3$ servers. Queue A has $N_A = 2$ servers, Queue B has $N_B = 3$ servers, and customers are processed at rates $N_A\mu$ and $N_B\mu$, respectively. A server switched from Queue B to Queue A would take θ units of time to arrive at Queue A.

μ. Figure 1 shows a schematic of such a queuing system having a pool of five servers distributed over two queues, each accommodating at most three servers.

Our objective is to dynamically switch servers between the two queues in order to minimize the total waiting time customers spend in both queues (excluding service time) over the course of a time horizon T, which could represent the length of a business day. While minimizing the total waiting time does not address the *variability* in waiting times, much of the work in the literature uses waiting time as an objective function. The use of other performance measures could be the focus of future research.

If a queue loses one or more servers, this loss is experienced immediately, while the queue receiving the additional servers experiences a lag of θ minutes before the servers come on duty, as they have to walk to the new queue and perhaps set up equipment at the new station. Hence, there is a tradeoff between responding to fluctuations in queue lengths and arrival rates, and maintaining sufficient service capacity.

There are many intuitive rules for guiding server switches in queuing systems having *homo-*

geneous arrival rates. For instance, one rule is to wait until Queue A has been emptied before switching any servers to Queue B. Another is never to switch a server from a long queue to a shorter queue. As we discuss later in this paper, these intuitive rules do not hold when arrivals are nonhomogeneous, so we require a different approach.

To reduce waiting time, we seek a time-varying allocation of servers to the two queues. However, rather than determining at the start of the day an allocation schedule that cannot be altered, we allow this allocation to be determined dynamically, at regularly spaced decision epochs. Decision epoch i begins at time t_i and has a duration of τ. The decision made at epoch i is a function of the current system state (number of people in the queues and current server allocation) and also knowledge of the future expected arrival rates. Thus, this is a *dynamic server allocation* problem where the goal is to determine optimally the conditions under which a server should be switched from one queue to another. The use of pre-specified decision epochs both reduces the computational complexity of the problem and is also realistic in many settings where managers cannot monitor queue lengths continuously.

Table 1 summarizes these important parameters.

3 Literature review

Having defined our problem, we will describe in this section how our work differs from that existing in the literature. Much queue control literature addresses the optimal routing of *customers* to parallel *servers* (e.g., Bell and Williams [2], Harrison and López [10] and Sethuraman and Squillante [28].) However, in certain queuing systems, load-balancing can be accomplished only by switching servers between queues.

There are also two additional characteristics that distinguish our work from the majority of work in the literature: time-varying customer arrival rates and non-zero switching times. While work in the literature often incorporates one of these characteristics at a time, none appears to address both simultaneously. We will provide below an overview of some of the relevant results; a more detailed literature review can be found in Martonosi [17].

The simplest version of the dynamic server allocation problem assumes a homogeneous arrival

Parameters	
$\lambda_A(t), \lambda_B(t)$	Time-dependent arrival rates to queues A and B at time t
N	Total number of servers available
N_{max_A}, N_{max_B}	Maximum number of servers that can ever be assigned to queues A and B
μ	Service rate
T	Duration of time horizon
t_i	Start time of decision epoch i, which is the time at which switches are allowed to occur
τ	Decision epoch interval at which switches are allowed to occur
θ	Time lag between the initiation of a switch and the arrival of the server at the new queue
α	Percent change in arrival rate from expected value in a given decision epoch (Defined in Section 4.2)
β	Probability that arrival rates will be a percentage α higher or lower than expected in a given decision epoch (Defined in Section 4.2)
L	Limited lookahead period explored by the dynamic program (Defined in Section 5.1)

Table 1: Glossary of key model parameters.

process and zero switching times when servers come on- or off-duty or are switched between queues. Much of this work focuses on non-anticipatory threshold policies (e.g. Moder and Phillips [20], and Squillante, *et al* [29]), and the $c\mu$-rule, which assigns the server to the fastest, most expensive queue first, under an assumption of linear holding costs (e.g. Baras, *et al* [1], Buyokkoc, *et al* [5], and Squillante, *et al* [29]). Chen and Yao [6] show that myopic optimization, minimizing the cost over the near future, yields a globally optimal solution over the entire time horizon in a fluid queuing network with feedback. We show in Section 4.1 that myopically choosing a server allocation over a limited time horizon is not optimal over the entire time horizon if switching times are non-zero.

There has also been extensive work on problems in which arrival rates are constant but switching times or costs incurred when a server switches between customer classes are non-zero. Such models are the subject of the survey by Rosa-Hatko and Gunn [27]. Only partial characterizations of optimal policies have been obtained for such problems. These policies again focus on non-anticipatory threshold policies (e.g. Berman and Larson [3], Hofri and Ross [11], and Koole ([12]-[13])), which will not be optimal when arrivals are nonhomogeneous. Duenyas and Van Oyen [7] develop a heuristic for minimizing expected holding costs that adapts the $c\mu$ policy to accommodate non-zero switching times. We modified this heuristic to accommodate nonhomogeneous arrivals, but as we mention in Section 6.3, this heuristic was not effective. Lu and Serfozo [15] saw that placing costs on such server changes causes them to occur less frequently. Hofri and Ross [11], and Liu, *et al* [14] show that optimal switching policies in polling systems are exhaustive: the queue being served must be depleted before the server can switch to the other queue. Liu, *et al* [14] show that a server will never be switched to a queue having a stochastically smaller queue length. These three findings seem intuitive, but we show in Section 6 that these intuitive rules break down if the policy is permitted to anticipate future changes to the arrival rate.

The next class of server allocation problems features time-varying arrival processes, but no switching times or costs. This work includes that of Mason, *et al* [18], Vandergraft [30], and Whitt [31], but does not reveal any general properties or algorithms that are applicable to the case of non-zero switching times.

It is the final category, covering nonhomogeneous arrivals *and* non-zero switching times, that sets the work in this paper apart from that in the literature. Fisher, *et al* [8], and Palmer and Mitrani ([22] and [23]) permit only one server to be switched at a time, whereas we allow multiple servers to be switched at a time. Moreover, their heuristics do not use knowledge of future arrival rate shifts in determining the current period's allocation. Similar problems have also been studied in the resource allocation and transportation scheduling literature. Powell [25] and related works by the same author examine a truck dispatching problem in a stochastic network assignment framework. However, their objective is to minimize transportation costs, not waiting time, and service times are assumed to be constant.

We now present our optimization models for this problem.

4 Problem formulations

We present first our most general framework for this queue control problem, and then discuss the simplifications that we must make to insure tractability.

At each decision epoch i starting at time t_i and ending at time $t_i + \tau$, we seek a pair of server allocations that minimizes the total expected waiting time incurred by all customers in the system or arriving to the system from time t_i through the end of the time horizon, T; we call this the "wait-to-go". The optimal allocation at decision epoch i depends not only on the time period and the expected future arrival rates but also on the *actual* state of the system at time t_i: the current queue lengths, Q_A and Q_B; the current server allocation (N_A, N_B); for queues with general arrival and service distributions, the times since the last customer arrivals to queues A and B, s_A and s_B; and the vectors $\vec{v_A}$ and $\vec{v_B}$ of time already spent in service by each customer in service at queues A and B. At epoch i and system state $S = (Q_A, Q_B, (N_A, N_B), s_A, s_B, \vec{v_A}, \vec{v_B})$ we denote the optimal server allocation, $(n_A(i, S), n_B(i, S))$. These variables are summarized in Table 2.

To find this optimal dynamic allocation, we use a dynamic programming (DP) formulation. The objective function that we would like to minimize is the total waiting time incurred by all customers who enter the system; that is, the sum of the customers' waiting times in queue. At any point in time t and current system state S, we denote by $W_t(S)$ the total waiting time incurred from time t onward to the end of the time horizon. This represents the sum total of the remaining

Variables	
Q_A, Q_B	Instantaneous queue lengths at Queues A and B at a given point in time.
N_A, N_B	Current allocation of servers to Queues A and B at a given point in time
s_A, s_B	Times since last customer arrivals to Queues A and B at a given point in time
$\vec{v}_A, \vec{v}_B$	Vectors of time already spent in service by the current customers in service at Queues A and B at a given point in time
$S = (Q_A, Q_B, (N_A, N_B), s_A, s_B, \vec{v}_A, \vec{v}_B)$	System state at a given point in time
$(n_A(i, S), n_B(i, S))$	Optimal number of servers allocated to Queues A and B at decision epoch i when the system is in state S
$W_t(S)$	Expected "wait-to-go", or total waiting time incurred by all customers in the system at time t or who will enter the system from time t onward to the end of the time horizon (time T), under an optimal allocation, starting in state S at time t
$w_t(S, (n_A, n_B), \theta)$	Waiting time incurred over the current decision epoch starting in state S at time t when allocation (n_A, n_B) is chosen and switches require θ time units to be completed
$\hat{S}_{t_i+\tau}(S, (n_A, n_B), \theta)$	State of the system at the end of the current decision epoch i with starting time t_i, duration τ, starting in state S, choosing allocation (n_A, n_B), and in which switches require θ time units

Table 2: Glossary of key model variables and recursive functions.

waiting times of any customers currently waiting in queue at time t and the waiting times of any customers who will enter the system after time t. (Our objective function is therefore $W_0(S_0)$ for a given initial system state S_0.) The total waiting time from time t onward, $W_t(S)$, can be broken into two terms. The first term represents the waiting time incurred in the current decision epoch, as a function of the server allocation (n_A, n_B) we choose and the time θ required for any switched servers to arrive at their new station. We represent this term by $w_t(S, (n_A, n_B), \theta)$. Once the allocation (n_A, n_B) is chosen and τ time units pass, we reach the end of the current decision epoch, and our system will have evolved to a new state, which we denote $\hat{S}_{t+\tau}(S, (n_A, n_B), \theta)$. The second term of $W_t(S)$ is therefore a recursive term representing the waiting time incurred from the start of the next decision epoch, at time $t + \tau$, to the end of the time horizon, as a function of the new server allocation decision made during that epoch. This "wait-to-go" is $W_{t+\tau}(\hat{S})$. For our purposes, the time t will always coincide with the start time t_i of a decision epoch i.

At any decision epoch i and state S, the optimal dynamic allocation is the choice of (n_A, n_B) achieving the minimum in

$$\begin{cases} W_T(.) &= 0 \\ W_{t_i}(S) &= \min_{(n_A,n_B)} E\left[w_{t_i}\left(S, (n_A, n_B), \theta\right) + W_{t_i+\tau}\left(\hat{S}_{t_i+\tau}(S, (n_A, n_B), \theta)\right)\right], \forall t_i < T. \end{cases} \quad (1)$$

This equation is known as Bellman's Equation. The expectation is taken with respect to the arrival and service distributions, and the minimization subject to $\max(N - N_{max_B}, 0) \leq n_A \leq \min(N_{max_A}, N)$ and $n_A + n_B = N$. From queuing theory, we know that the total waiting time in the system in people-minutes, excluding time spent in service, is the integral of the instantaneous queue length over time. So, for each decision epoch i starting at time t_i and ending at time $t_i + \tau$, $w_{t_i}(S, (n_A, n_B), \theta)$ is the sum of the instantaneous queue lengths at the two queues, integrated over the time interval $(t_i, t_i + \tau)$.

With all of the possible combinations of queue lengths, allocations and residual entry and service time vectors, $\vec{s}$ and $\vec{v}$, there is a large set of possible states at each decision epoch. The formulation in Eq. (1) is rendered even more complicated by the large number of possible random trajectories of the system that must be considered in order to compute the expected values of w_t and $W_{t+\tau}$.

To get around these difficulties, we use *fluid queues*.

4.1 Deterministic fluid queue

A fluid queue is one in which arriving customers are not considered as discrete individuals but as a fluid flowing continuously into the system, much like fluid passing through a pipe into a bucket with a hole in its bottom. The rate at which the fluid passes through the pipe into the bucket is the arrival rate of the customer fluid to the queue, the amount of fluid in the bucket is the queue length, and the rate at which fluid leaves the bucket through the hole is the service rate. In a fluid queue having k servers, the servers do not work individually to serve customers, but rather work together at a combined rate of $k\mu$ (i.e., the hole in the bucket is larger). Fluid queues are appropriate models for high-volume queuing systems in which the discrete effects of individual customers arriving and departing are small relative to the volume of the system. Just as we can analyze the flow of water through a pipe without distinguishing individual water molecules, we can understand the behavior of high-volume queuing systems without distinguishing individual customers. Fluid queues have been used to model many types of queues such as telecommunication queues (e.g., Mandelbaum, *et al* [16]), traffic intersections (e.g., Panayiotou, *et al* [24]) and call centers (e.g., Ridley, *et al* [26]). In addition to being asymptotically accurate representations of queuing systems that are operating near maximum capacity, Panayiotou, *et al* [24] and the references therein argue that fluid queues are suitable approximations of many queuing systems for the purposes of optimization.

Using a fluid queue as an approximation can render tractable an otherwise intractable optimization problem. For example, in a fluid queue, we do not need to keep track of the residual inter-entry times or the residual service times for each customer in service $(s_A, s_B, \vec{v_A}, \vec{v_B})$, greatly reducing our state space. Moreover, because arrivals and services occur at a continuous rate, we do not need to consider customers individually when calculating total customer waiting times in a fluid queue. Instead, we can rely on easily computable quantities from calculus.

First, we define *cumulative arrivals* and *cumulative services* to be the total number of arrivals or services that have occurred up to a given time. As discussed by Newell [21], the instantaneous queue length at time t is the difference between the cumulative arrivals and cumulative services up to time

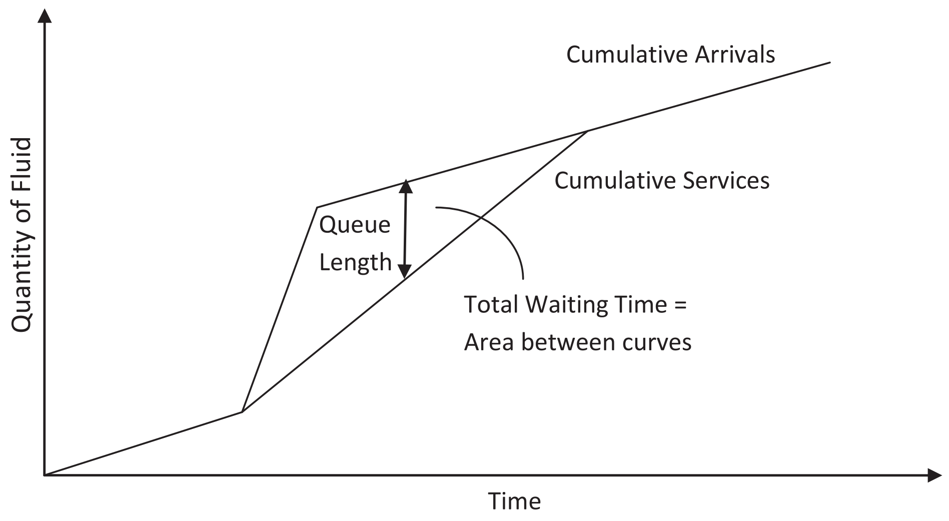

Figure 2: Cumulative arrivals (upper curve) and services (lower curve) in a fluid queue with piecewise constant arrival rates. If the total service rate exceeds the arrival rate before a queue has formed, then fluid is processed immediately as it enters and no queue forms. If the arrival rate exceeds the service rate, then a queue develops that can be reduced only when the arrival rate falls below the total service rate.

t. The total waiting time in the system is the integral of the instantaneous queue length, which is the area between these two curves. In a fluid queue, if the time-varying arrival rates are piecewise constant, then the cumulative arrivals curve over time is a non-decreasing, piecewise linear function with breakpoints coinciding with changes in the arrival rates. For our purposes, we will assume that changes in the arrival rate coincide with decision epochs, so that we can assume that the arrival rate remains constant during a given decision epoch. Likewise, the cumulative services curve over time is also a non-decreasing, piecewise linear function with breakpoints occurring whenever servers are switched, or whenever the queue is depleted so that the service rate equals (and changes with) the arrival rate. Therefore, the total system waiting time is just the area between two piecewise linear functions, which can be computed using simple geometry.

This is illustrated in Figure 2. If the service rate at a queue exceeds the arrival rate when no queue has formed, then fluid is processed linearly at the arrival rate (fluid cannot be processed faster than it appears at the queue). Because it is processed as soon as it enters, no queue is formed in this case, and the total waiting time remains zero. On the other hand, if the arrival rate is greater than the total service rate, then a queue grows linearly with time at a rate equal to their difference. Whenever a queue has formed, fluid is processed at the maximum service rate. If this maximum service rate exceeds the arrival rate, then the queue is depleted linearly with time at a rate equal to their difference; otherwise the queue continues to grow.

In this simplest fluid queue framework, the queue lengths evolve deterministically for a given set of arrival and service rates and a given schedule of server allocations over the course of the day. Therefore, the optimal server allocation for each decision epoch need not be computed dynamically in response to system evolution but can instead be determined prior to the start of the time horizon. We call the optimal allocation for this completely deterministic fluid queue scenario the *deterministic benchmark allocation.* The optimal allocation for this scenario is the type of allocation that typically comes to mind in the context of staff scheduling: it designates at the start of the day how many employees are to be assigned to each post for each decision epoch. We assume the deterministic benchmark allocation reflects current practice in most queuing systems because staffing allocations are generally created assuming a deterministic evolution of the system.

To find the best *deterministic benchmark allocation*, we solve the DP given in Eq. (1), where the expectation can be omitted due to the deterministic evolution of the system.

A reasonable question is whether a greedy allocation, one that minimizes the wait incurred over only the current decision epoch, $w(t, S)$, might be optimal over the entire time horizon. If this were the case, then a dynamic programming framework would not be needed, and the allocations could be selected on the basis of the current period alone without considering the future. However, a counterexample shows that this is not the case.

Example. Consider two fluid queues, A and B, of lengths $Q_A = 75$ and $Q_B = 15$ at time t for which, for simplicity, there will be no future arrivals through the end of the time horizon at time $t + 90$. Let μ, the service rate, be equal to 0.5 units per minute, decisions occur every 30 minutes, and switches require 15 minutes to be completed. Suppose that there are $N = 2$ servers and both are allocated to Queue B immediately prior to time t. Our state S at time t is thus $(75, 15, (0, 2))$.

Consider a greedy allocation that minimizes only the wait to be incurred over the immediate period, $(t, t+30)$. There are three possible allocations to consider: (0,2), in which all of the servers remain at Queue B; (1,1), in which one server is switched from Queue B to Queue A, leaving Queue B immediately and arriving to Queue A at time $t + 15$; and (2,0), in which both servers leave Queue B immediately to arrive at Queue A at time $t + 15$.

Of the three choices, allocation $(0, 2)$ minimizes the waiting time over the first period. This allocation depletes Queue B within fifteen minutes while all 75 customers in Queue A continue to wait for the entire thirty minute period. The waiting time is given by the area between the cumulative arrivals and cumulative services curves over the interval from t to $t + 30$ in Figure 3: 2250 person-minutes at Queue A plus 112.5 person-minutes at Queue B, for a total of 2362.5 person-minutes.

Following this allocation, the system state S at time $t + 30$ will be $(75, 0, (0, 2))$, and therefore both servers must be switched to Queue A at time $t + 30$ and arriving at time $t + 45$. Queue A then incurs 2137.5 person-minutes of wait time over the time interval $(t + 30, t + 60)$ and 1350 person-minutes over the interval $(t + 60, t + 90)$, as shown in Figure 3. As Queue B experiences no further arrivals, its additional wait time is zero person-minutes. Thus, the greedy allocation which achieves the minimal waiting time in the first period achieves a total waiting time over the three periods of 5850 person-minutes.

However, solving the dynamic program reveals that the globally optimal allocation is to switch one server from Queue B to Queue A at time t and then to select allocation (2,0) at times $t + 30$ and $t + 60$. By computing the areas between the cumulative arrivals and cumulative services curves in Figure 4, we see this yields a total wait of only 5062.5 person-minutes. That is, by incurring a slightly higher waiting time at Queue B in the first period (225 person-minutes as compared to 112.5 person-minutes), the system is better positioned to work off Queue A in subsequent periods, thus minimizing total wait. $\square$

Thus, it is insufficient to determine server allocations based on a myopic view of the current period alone, and the aforementioned dynamic program must be solved to find the optimal deterministic benchmark allocation.

We have just described the deterministic benchmark allocation that is optimal in a completely deterministic fluid queue, but which does not permit the allocations to change in response to

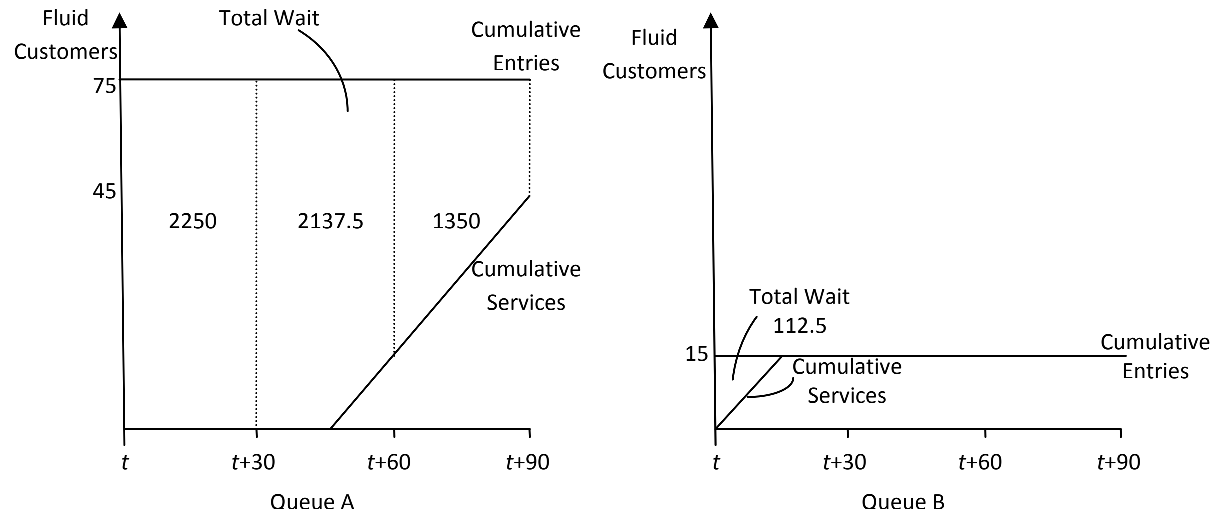

Figure 3: System evolution under greedy server allocations: (0,2) at time t, (2,0) at time $t+30$, and (2,0) at time $t+60$. Although the waiting time during the period $(t, t+30)$ is minimized (2362.5 person-minutes), the total wait over $(t, t+90)$ is not (5850 person-minutes), as demonstrated by the optimal allocation shown in Figure 4.

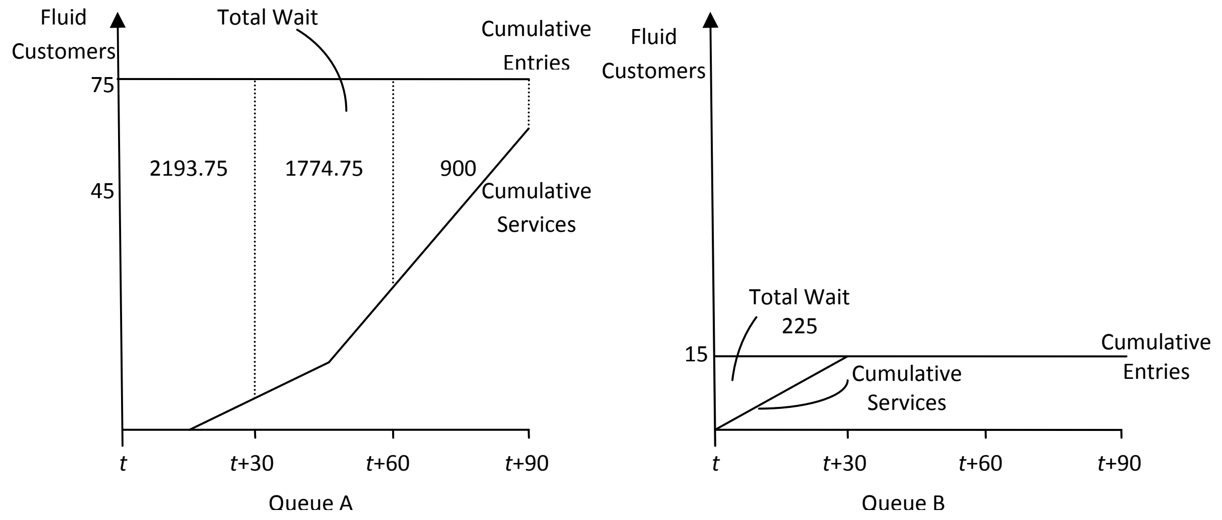

Figure 4: System evolution under optimal server allocations: (1,1) at time t, (2,0) at time $t+30$, and (2,0) at time $t+90$. The waiting time in the first period $(t, t+30)$ is not as small (2418.75 person-minutes) as under the greedy allocation, but despite this initially suboptimal allocation, the total wait over $(t, t+90)$ is minimized (5062.5 person-minutes).

system stochasticity. Because our objective is to evaluate the improvement offered by reallocating servers dynamically as opposed to adhering to a predetermined allocation, we will use this original deterministic benchmark allocation as a baseline to which dynamic allocations on stochastic variants of the fluid queue model will be compared.

We can introduce stochasticity into our fluid queue framework by assuming that the arrival rate for each decision epoch varies randomly about its mean, or that the service rate experienced by each unit of flow in the system varies randomly. This gives us two scenarios to compare to the deterministic benchmark scenario: a scenario with randomized arrival rates, and a scenario with randomized service times. These are described in the following sections.

4.2 Fluid queue with randomized arrival rates

Our first stochastic variant within the fluid queue framework considers arrival rates that, independently in each decision epoch for each queue, randomly deviate from their originally estimated values according to a simple probability distribution. We assume that if the arrival rate in a given period at a given queue is estimated to be λ, then the actual arrival rate will be $(1-\alpha)\lambda$ with probability β, $(1+\alpha)\lambda$ with probability β and λ with probability $1-2\beta$, where $0 \leq \alpha \leq 1$, and $0 \leq \beta \leq 0.5$. This simple probability distribution for the arrival rates reflects a scenario where the estimated arrival rates are not very accurate, or where the number of customers arriving at the queue changes by a fixed proportion. (In an airport security context, for instance, this could arise due to gate changes or flight cancelations.) High values of α and β indicate greater uncertainty about the arrival rates. Note that once the arrival rate for the current decision epoch has been selected from this distribution, the arrival flow rate is fixed at this rate during the decision epoch, as per the usual fluid queue framework.

Because either queue's arrival rate can be $(1-\alpha)\lambda_t$, λ_t or $(1+\alpha)\lambda_t$ at time t, there are nine random outcomes to consider, and the dynamic program of Eq. (1) becomes

$$\left\{\begin{array}{lcl}
W_T(.) & = & 0 \\
W_t(S) & = & \min_{(n_A,n_B)} \beta^2 \left[w_t\left(S,(n_A,n_B),\theta,(1-\alpha)\lambda_{A,t},(1-\alpha)\lambda_{B,t}\right) \right. \\
& & +W_{t+\tau}\left(\hat{S}_{t+\tau}(S,(n_A,n_B),\theta,(1-\alpha)\lambda_{A,t},(1-\alpha)\lambda_{B,t})\right) \\
& & +w_t\left(S,(n_A,n_B),\theta,(1-\alpha)\lambda_{A,t},(1+\alpha)\lambda_{B,t}\right) \\
& & +W_{t+\tau}\left(\hat{S}_{t+\tau}(S,(n_A,n_B),\theta,(1-\alpha)\lambda_{A,t},(1+\alpha)\lambda_{B,t})\right) \\
& & +w_t\left(S,(n_A,n_B),\theta,(1+\alpha)\lambda_{A,t},(1-\alpha)\lambda_{B,t}\right) \\
& & +W_{t+\tau}\left(\hat{S}_{t+\tau}(S,(n_A,n_B),\theta,(1+\alpha)\lambda_{A,t},(1-\alpha)\lambda_{B,t})\right) \\
& & +w_t\left(S,(n_A,n_B),\theta,(1+\alpha)\lambda_{A,t},(1+\alpha)\lambda_{B,t}\right) \\
& & \left. +\, W_{t+\tau}\left(\hat{S}_{t+\tau}(S,(n_A,n_B),\theta,(1+\alpha)\lambda_{A,t},(1+\alpha)\lambda_{B,t})\right)\right] \\
& & +\beta(1-2\beta)\left[w_t\left(S,(n_A,n_B),\theta,(1-\alpha)\lambda_{A,t},\lambda_{B,t}\right) \right. \\
& & +W_{t+\tau}\left(\hat{S}_{t+\tau}(S,(n_A,n_B),\theta,(1-\alpha)\lambda_{A,t},\lambda_{B,t})\right) \\
& & +w_t\left(S,(n_A,n_B),\theta,(1+\alpha)\lambda_{A,t},\lambda_{B,t}\right)+W_{t+\tau}\left(\hat{S}_{t+\tau}(S,(n_A,n_B),\theta,(1+\alpha)\lambda_{A,t},\lambda_{B,t})\right) \\
& & +w_t\left(S,(n_A,n_B),\theta,\lambda_{A,t},(1-\alpha)\lambda_{B,t}\right)+W_{t+\tau}\left(\hat{S}_{t+\tau}(S,(n_A,n_B),\theta,\lambda_{A,t},(1-\alpha)\lambda_{B,t})\right) \\
& & \left. +w_t\left(S,(n_A,n_B),\theta,\lambda_{A,t},(1+\alpha)\lambda_{B,t}\right)+W_{t+\tau}\left(\hat{S}_{t+\tau}(S,(n_A,n_B),\theta,\lambda_{A,t},(1+\alpha)\lambda_{B,t})\right)\right] \\
& & +(1-2\beta)^2\left[w_t\left(S,(n_A,n_B),\theta,\lambda_{A,t},\lambda_{B,t}\right)+W_{t+\tau}\left(\hat{S}_{t+\tau}(S,(n_A,n_B),\theta,\lambda_{A,t},\lambda_{B,t})\right)\right].
\end{array}\right. \tag{2}$$

Because the arrival rates deviate randomly from their expected values from one decision epoch to the next, the evolution of the system is no longer deterministic. Solving this equation will yield a dynamic server allocation policy that depends not just on the decision epoch but on the specific system states encountered as the system evolves stochastically. We will later compare the waiting times experienced under this dynamic allocation to those experienced under the deterministic benchmark allocation described above.

4.3 Fluid queue with random service times

Our second stochastic variant to the fluid queue keeps arrival rates deterministic but considers instead randomness in the service process. Until now, we have been assuming that each unit of flow, representing a customer, has a constant service time equal to $1/\mu$, but in reality, different customers tend to require different service times. We can model this in our fluid queue framework by assigning to each unit of fluid, j, in queue i an exponentially distributed service time X_j with mean $1/\mu$ minutes, which represents the total amount of "work" that must be done on that unit of flow. That work is divided over all N_i servers on-duty at that queue, such that the unit of fluid is processed at a uniform rate equal to N_i/X_j. Thus, we maintain the continuous fluid form of the model while stochastically changing, on a unit-by-unit basis, the rate at which the fluid is processed.

We can use Eq. (1) to find the optimal dynamic allocation for the case of random service times, where the expectation is with respect to the random service times, and augmenting the system state S from the deterministic fluid queue to include the remaining service time of the unit of fluid currently being processed. The waiting times under this dynamic allocation will later be compared to those yielded by the deterministic benchmark allocation described above.

We have now defined the three model frameworks we will consider: deterministic fluid flows, which yield a deterministic benchmark allocation; fluid flows with randomized arrival rates, which might benefit from a dynamic solution; and fluid flows with randomized service times, which might benefit from a dynamic solution.

5 Approximate DP solution techniques

For the non-deterministic frameworks, solving Eq. (1) for the dynamic server allocations quickly becomes computationally intractable. In this section, we describe approximations to facilitate solving these dynamic programs.

Although the fluid queue formulations allow us to ignore the individuality of customers, we must still keep track of, at each time point, the amount of fluid in each queue (which might be fractional) and the current server allocation. This causes the state space in Eq. (1) to become prohibitively large. Furthermore, the nonstationarity of the entry processes over the course of the day forces the wait-to-go function to be time-dependent, preventing steady-state conditions which would simplify the system of equations. For instance, if decisions to switch servers can be made every half hour over an 18-hour day, even the deterministic model becomes intractable, and the problem is even harder if switching decisions are made more frequently.

Thus, to solve the models described earlier, we must resort to approximate dynamic programming techniques. (A detailed discussion of such techniques is provided in chapter six of Bertsekas [4]).

5.1 Aggregated state space with limited lookahead

For all of the models considered (deterministic and stochastic), we rely upon state space aggregation and limited lookahead. In many queue control situations, queue lengths are generally not monitored precisely. A manager overseeing parallel queues might observe "roughly 20" people waiting in line, but is unlikely to take the time to count them exactly when deciding whether or not to reallocate servers between queues. Thus, for our model, we will aggregate queue lengths into multiples of ten when choosing server allocations. Note that this is a substantial reduction in the state space because fluid queue frameworks typically yield fractional queue lengths.

We also limit the "lookahead" period, the number of future decision epochs considered during the current decision epoch, to reflect the fact that decisions made at 6:30AM are unlikely to cause a significant impact on the state of the system at 4:00PM, for instance. This helps to truncate the exponential growth of the solution space. We let L be the lookahead period explored into the future by the DP.

We can modify Eq. (1) to incorporate these approximations. Previously, our system state was the vector $S = (Q_A, Q_B, (N_A, N_B))$. In an aggregated state space, our system state is the vector $S = (\tilde{Q}_A, \tilde{Q}_B, (N_A, N_B))$, where $\tilde{Q}_A$ and $\tilde{Q}_B$ are the values of the queue lengths rounded to the nearest multiple of ten. To incorporate a limited lookahead, we need two time arguments for the wait-to-go function, $W_t^{\hat{t}}$: $\hat{t}$ is the global clock measuring the starting time of our limited lookahead, and t is the current time on the clock. If $t \geq \hat{t} + L$, then $W_t^{\hat{t}} = 0$. Bellman's Equation becomes:

$$\begin{cases} W_t^{\hat{t}}(S) &= 0, \text{ for } t \geq \hat{t} + L \\ W_T^{\hat{t}}(S) &= 0 \\ W_t^{\hat{t}}(S) &= \min_{(n_A, n_B)} w_t\left(S, (n_A, n_B), \theta\right) + W_{t+\tau}^{\hat{t}}\left(\hat{S}_{t+\tau}\left(S, (n_A, n_B), \theta\right)\right). \end{cases} \tag{3}$$

As before, we let T be the end of the time horizon, τ be the decision epoch, θ be the switching time, and (n_A, n_B) be the server allocation to the two queues. The aggregated system state (e.g. queue lengths of each queue, rounded to multiples of ten) at the current time and after the system has evolved over the decision epoch are S and $\hat{S}$, respectively. The wait incurred in the current period is given by $w_t\left(S, (n_A, n_B), \theta\right)$.

5.2 Approximating expected values

In the deterministic and randomized arrival rate models, the arrivals and services are deterministic within each period (in the latter case, once the arrival rate for the period has been revealed). So state aggregation with a limited lookahead period is a sufficient approximation for solving these models in reasonable time. For the randomized service time model, however, calculating the expected wait-to-go is significantly more difficult because each customer requires its own randomized service time. Therefore, we require an additional approximation, as we describe here.

In the case of random service times, each unit of flow is assigned its own exponentially distributed service time, and the expected value in Eq. (1) must be taken over infinitely many random trajectories. To address this, we implement two different heuristics, both involving simulation, rather than exact computation, of the expected value.

5.2.1 A hybrid allocation

The first of the heuristics we consider is a "hybrid" of a stochastic dynamic program and a deterministic dynamic program, in which the allocation at each state and stage is made assuming the system evolves deterministically, but once the allocation is chosen, the actual evolution of the system is stochastic. This allows the system to visit states not visited by the original deterministic DP and from those states to select allocations different from those in the original deterministic benchmark allocation. (This is similar in spirit to the BIGSTEP method of Harrison [9].) Rather than computing the expected value in Eq. (1) exactly, we simulate the evolution of the system. For iterations $k = 1...K$ ($K = 500$ in our simulations) having stochastic trajectory ω_k, we compute $W_0(S_0)_k$ from an initial state S_0 recursively, as follows:

- **Function** $W_t(S)_k$:

1. If $t = T$, return 0. Otherwise,

2. **Deterministic allocation step**

   Find

$$\overline{(n_A, n_B)}_{S,t} \quad = \quad \arg \min_{(n_A,n_B)} \tilde{w}_t \left(S, (n_A, n_B), \theta\right) + \tilde{W}^t_{t+\tau}\left(\hat{S}_{t+\tau}(S, (n_A, n_B), \theta)\right), \quad (4)$$

   where $\tilde{w}_t \left(S, (n_A, n_B), \theta\right)$ is the waiting time incurred during time interval $(t, t+\tau)$ in a deterministic fluid queue, and

$$\begin{cases} \tilde{W}^{\hat{t}}_t(S) & = & 0, \text{ for } t \geq \hat{t} + L \\ \tilde{W}^{\hat{t}}_t(S) & = & \min_{(n_A,n_B)} \tilde{w}_t \left(S, (n_A, n_B), \theta\right) + \tilde{W}^{\hat{t}}_{t+\tau}\left(\hat{S}_{t+\tau}\left(S, (n_A, n_B), \theta\right)\right) \end{cases} \quad (5)$$

   is the deterministic fluid flow problem with limited lookahead L.

3. **Stochastic evolution step**

   Return

$$W_t(S)_k \quad = \quad w_t\left(S, \overline{(n_A, n_B)}_{S,t}, \theta, \omega_k\right) + W_{t+\tau}\left(\hat{S}_{t+\tau}\left(S, \overline{(n_A, n_B)}_{S,t}, \theta, \omega_k\right)\right)_k, \quad (6)$$

   where $w_t\left(S, \overline{(n_A, n_B)}_{S,t}, \theta, \omega_k\right)$ is the waiting time incurred during time interval $(t, t+\tau)$ under stochastic evolution ω_k and using the allocation $\overline{(n_A, n_B)}_{S,t}$ found in Step 2; and $W_{t+\tau}\left(\hat{S}_{t+\tau}\left(S, \overline{(n_A, n_B)}_{S,t}, \theta, \omega_k\right)\right)_k$ is the "wait-to-go" from time $t+\tau$ to the end of the time horizon under stochastic evolution ω_k and using the allocation $\overline{(n_A, n_B)}_{S,t}$ found in Step 2.

The average total waiting time is estimated by $W_0(S_0) = \frac{1}{K}\sum_k W_0(S_0)_k$. Step 2 is where the allocation is selected based on the deterministic model, and Step 3 is where the system evolves stochastically, according to ω_k, resulting in a new state $\hat{S}$ from which to recurse.

5.2.2 Nearest neighbor heuristic

The second approximation heuristic used in the random service times framework is a nearest neighbor heuristic. Ideally we would like to allow the allocations chosen by the DP in Step 2 above to consider the stochastic evolution of the system. We can simulate the stochastic evolution of the system in Step 2, but in doing so, we are performing several times (in our simulations, 100 iterations) the very step that takes the longest to evaluate. Thus, we must restrict the number of possible allocations we are willing to consider in the minimization. Because the optimal stochastic allocation in decision epoch i is likely to be close to the original deterministic benchmark allocation, $(n_A(i), n_B(i))$, we choose an allocation from a set of nearest neighbors to this allocation.

We modify Step 2 above as follows: rather than taking the $\arg\min$ over all feasible allocations (n_A, n_B), we select the $\arg\min$ over the set of allocations that differ by at most one switched server from the benchmark allocation. For a deterministic benchmark allocation $(n_A(i), n_B(i))$, this set is

$$\{(n_A(i)-1, n_B(i)+1), (n_A(i), n_B(i)), (n_A(i)+1, n_B(i)-1)\}.$$

From this set we select the allocation that minimizes the *average* limited lookahead wait-to-go over several (100) simulation runs. Once an allocation has been chosen, we continue with Step 3 as before, simulating the system evolution until the next decision epoch.

6 Results

In this section, we apply the approximate dynamic programming approach on the three modeling frameworks described above.

We use two arrival rate data sets, which are described below: synthetic arrival rate data and real arrival rate data from a pair of airport security checkpoints from Boston's Logan International Airport. The synthetic arrival rates demonstrate the potential benefit of using dynamic server allocation to reduce waiting times under ideal circumstances. By "ideal" we mean that we expect dynamic reassignment of servers to be most beneficial when peak periods in one queue correspond to slow periods in the other. The second set of data, from Logan Airport, give an idea of how well these techniques might work in a real queuing system. Logan Airport is a high-volume airport, handling over 170,000 departing flights in 2009, which corresponds to over 12 million departing passengers for the year, or roughly 35,000 passengers passing through security per day [19].

Throughout this section, we refer back to the arrival and service scenarios described earlier. The "deterministic benchmark allocation" refers to the allocation described in Section 4.1 for the deterministic fluid queue. It is solved once prior to the start of the day and thus does not dynamically adjust to the current system state. A "dynamic allocation" is a server allocation that is permitted to be altered during the course of the day in response to queue imbalances. For the case of randomized arrival rates (Section 4.2), the dynamic allocation is that given by Eq. (2). For the case of random service times (Section 4.3), the hybrid dynamic allocation is that given in Section 5.2.1, and the nearest neighbor dynamic allocation is that given in Section 5.2.2. All four make use of aggregated state space and limited lookahead, as described in Section 5.1.

6.1 Synthetic Data

To test these models, we start by demonstrating their potential on an "ideal" synthetic data set: Because our model holds the total number, N, of servers on-duty constant, we hypothesize that dynamic server allocation might be most useful in systems where the total demand on the system is fairly constant but a peak period at one queue corresponds to a slow period at the other queue and vice versa. We will therefore construct a synthetic data set that has this property to see if our methods work under ideal conditions.

The time horizon is a 13.5 hour day (from 5 AM to 6:30 PM). Decisions to switch servers between queues can be made every 30 minutes; that is, the decision epochs (τ) are 30 minutes long. Although more precise queue control could be achieved by permitting more frequent switches, this increases the computation time of the problem considerably. Even decision epochs every 15 minutes were found computationally intractable for the stochastic versions of the problem. However, 30 minutes is a realistic decision epoch in many applications. For instance, at San Francisco International Airport, where the practice of switching servers between queues was observed, the official monitoring the queues over video feed was primarily identifying security breaches, rather than closely monitoring

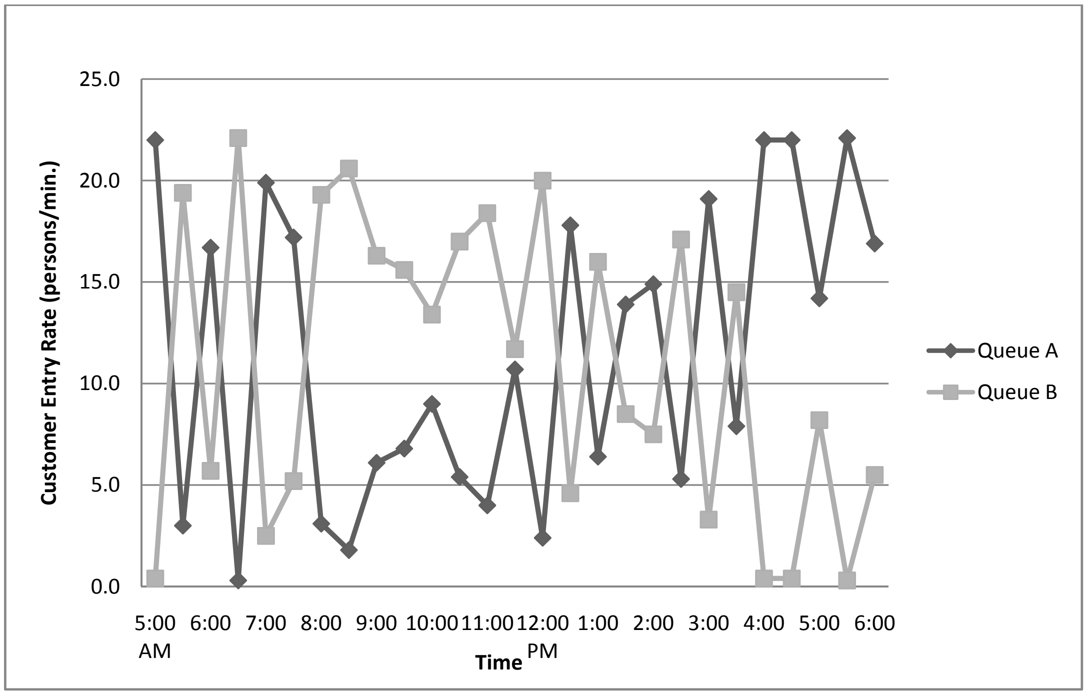

Figure 5: Customer arrival rates (in customers per minute) to Queues A and B. (Synthetic arrival rates.)

queue lengths. Moreover, it makes sense to require decision epochs be longer than the switching time, θ, which we set equal to 0, 5, 10, 15, or 30 minutes.

So that the waiting times observed on this synthetic data set will be roughly comparable to those observed for the Logan Airport data (see Section 6.2), the service rate is chosen to be 2.8 customers per minute, with a total pool of $N = 10$ servers. The total arrival rate over the two queues in each period is 22.4 customers per minute (corresponding to an 80% systemwide load). The arrival rate λ_A at Queue A for each 30-minute period is chosen uniformly from $(0, 22.4)$, and the arrival rate λ_B at Queue B for the same period is set equal to $22.4 - \lambda_A$. This ensures that the total system load stays constant over the day and that periods with a high arrival rate at Queue A correspond to periods with a low arrival rate at Queue B. These arrival rates are shown in Figure 5 for the period from 5:00 AM to 6:30 PM.

Table 3 summarizes the results of applying our dynamic allocation methods to the synthetic data, as compared to the deterministic benchmark allocation. The columns labeled Deterministic Benchmark Allocation give the average waiting times (in minutes per customer) and the total num-

	Deterministic Benchmark Allocation (Section 4.1)		Random Arrival Rates (Section 4.2)						Random Service Times (Section 4.3)				
			$\alpha = 0.1$, $\beta = 0.1$			$\alpha = 0.3$ $\beta = 0.3$							
			Wait (Min.)			Wait (Min.)			Wait (Min.)			Wait (Min.)	
θ	Wait (Min.)	No. of Switches	Bench-mark	Dyn-amic	% Impr.	Bench-mark.	Dyn-amic.	% Impr.	Bench-mark	Dynamic (Hybrid)	% Impr.	Dynamic (Neigh.)	% Impr.
0	0.00	56	0.08	0.00	100.0	2.20	0.69	68.6	0.32	**0.29**	9.4	**0.12**	62.5
5	0.43	85	0.53	0.50	5.7	3.83	2.23	41.8	0.70	**0.68**	2.8	**0.64**	8.6
10	1.79	76	2.06	1.93	6.3	6.98	3.95	43.4	2.24	**2.09**	6.7	**2.07**	7.6
15	2.72	27	2.84	2.89	-1.8	6.50	4.95	23.8	3.02	**2.98**	1.3	**2.92**	3.3
30	3.77	9	3.84	4.28	-11.4	7.55	6.78	10.2	4.06	4.03	0.7	**4.01**	1.2

Table 3: Results of dynamic server allocation methods applied to synthetic arrival rate data, by switching time, θ. The columns labeled Deterministic Benchmark Allocation give the average waiting times (in minutes per customer) and the total number of servers switched under deterministic benchmark allocation applied to the deterministic fluid queue. The columns labeled Random Arrival Rates give the average waiting times for the deterministic benchmark allocation ("Benchmark") and the dynamic allocation ("Dynamic") applied to the randomized arrival rate fluid queues with ($\alpha = 0.1$, $\beta = 0.1$) and ($\alpha = 0.3$, $\beta = 0.3$), and the percentage improvement of the dynamic allocation over the deterministic benchmark allocation. The columns labeled Random Service Times give average waiting times for the deterministic benchmark allocation ("Benchmark"), the hybrid dynamic allocation ("Hybrid") and the nearest neighbor dynamic allocation ("Neigh.") applied to the randomized service time fluid queues, as well as the percentage improvement the dynamic hybrid and nearest neighbor allocations have over the deterministic benchmark allocation. Numbers in boldface indicate a statistically significant difference from the benchmark deterministic allocation at a 95% confidence level).

ber of servers switched under the deterministic benchmark allocation applied to the deterministic fluid queue for each value of the switching time θ. As expected, when the switching time increases, average waiting times increase. We might also expect the number of switches taking place to decrease as a function of θ because switches are, in a sense, more costly. This is not the case, in fact. The number of switches actually *increases* for a small increase in θ from zero to five minutes. When a switch *must* occur to accommodate a sharp change in the arrival rates, the higher switching time occasionally causes such a long queue to form at the recipient queue that the system must switch even more servers to compensate. This is in contrast to the findings of Lu and Serfozo [15] that for the case of constant queue arrival rates, as the switching costs increased, the number of switches decreased monotonically. We can now compare the performance of this deterministic benchmark allocation to that of our dynamic allocations.

In Table 3, the columns labeled Random Arrival Rates give the average waiting times for the deterministic benchmark allocation ("Benchmark") and the dynamic allocation ("Dynamic") applied to the randomized arrival rate fluid queues with ($\alpha = 0.1$, $\beta = 0.1$) and ($\alpha = 0.3$, $\beta = 0.3$) from Section 4.2, and the percentage improvement of the dynamic allocation over the deterministic benchmark allocation. We note, as expected, that the average waiting time under the original deterministic benchmark allocation is higher when applied to the stochastic system than when applied to the deterministic system, particularly in the high variance case ($\alpha = 0.3$, $\beta = 0.3$) where the stochasticity alone adds several minutes to the average waiting time. Next we see that in the system where the variance in the arrival rates is low ($\alpha = 0.1$, $\beta = 0.1$), dynamic allocation offers a small reduction in the average waiting time over the original deterministic benchmark allocation and occasionally performs worse due to the approximate nature of the DP. However, in a system with a high level of uncertainty about the actual arrival rates ($\alpha = 0.3$, $\beta = 0.3$), dynamic allocation yields at least a 10% reduction in the average waiting time over the original deterministic benchmark allocation, and when the switching time is low, the reduction is even more substantial. This suggests that the greater the uncertainty in the arrival rates, the more the system can benefit from determining allocations dynamically rather than adhering to a pre-determined schedule based on unreliable information.

The columns of Table 3 labeled Random Service Times compares the benchmark deterministic allocation ("Benchmark") to the hybrid ("Hybrid") and nearest neighbor ("Neigh.") dynamic allocations applied to the random service time fluid queues, over 500 simulation trials. We note again the price of stochasticity: the introduction of random service times causes the average wait to increase, even when the same deterministic benchmark allocation is used. Here, the increase is significantly smaller, only 0.32 minutes or roughly twenty seconds, than in the case of randomized arrival rates. We see further that the dynamic allocations often yield a statistically significant decrease in the average waiting time, and occasionally yield a reduction of more than 5%, particularly when switching costs are low. The hybrid allocation offered reductions in waiting time between 1% and 10%, and the nearest neighbor heuristic performed better, with reductions of 1% to 62%. This suggests that taking the stochastic nature of the system into account when selecting an allocation (as the nearest neighbor heuristic does) is more valuable than exploring all possible allocations (as the hybrid heuristic does). Lastly, we see that the benefits of a dynamic allocation in a system with random service times are not as pronounced as in the case with highly variable arrival rates. The arrival rate disruptions we considered in the previous section affected all customers scheduled to enter during each period. Such widespread change to the arrival rate should naturally cause significant increases in waiting times if an allocation is not robust to these changes. By contrast, when a system's uncertainty is at the level of individual customers' service times, the stochastic fluctuations may roughly cancel out over a decision epoch.

6.2 Boston Logan International Airport data

We have just demonstrated that dynamic server allocation, using approximate dynamic programming, can help achieve reductions in average customer waiting times at a pair of queues with "ideal" customer arrival rates. In this section, we present a case study of security checkpoint queuing data from Boston Logan International Airport to examine the performance of these dynamic allocation heuristics on real queue data.

The data provided by Massport include throughput and total number of lanes open at each security checkpoint at Logan airport, on an hourly or half-hourly basis for January 18, 2005. On this date, Logan Airport had five terminals, each having at least one security checkpoint serving a subset of gates within the terminal. Each checkpoint has a certain number of lanes equipped with an x-ray bag scanner for carry-on luggage and a metal detector gate through which customers walk. We use the term *server* to refer to an open lane including equipment (x-ray machine and metal detector) and manpower (employees to operate the equipment, direct customers through the lane and conduct searches). For example, at checkpoint E2, while there is capacity for up to seven open lanes, at any given time there might be only five "servers" on-duty, corresponding to five sets of x-ray machines, metal detectors and screening employees.

In our framework, Queues A and B are two airport security checkpoints serving different sets of aircraft departure gates. Our analysis uses the data of Terminals C and E, treating Terminal C's three checkpoints (having a total of eleven possible screening lanes) as one combined checkpoint, and treating Terminal E's two checkpoints (having a total of nine lanes) as one. We choose these two terminals because the total number of checkpoint lanes open (which determines the size of our server pool, N) remained fairly constant across the day. According to the data from Logan airport, from 5:00 AM to 6:30 PM on January 18, 2005, the total number of checkpoint lanes open at Terminals C and E was roughly 10, so we use $N = 10$ in our trials.

We estimate the arrival rate, $\lambda_i(t)$ for any terminal i from the throughput information provided by Massport. We let $\lambda_i(t)$ be equal to the average throughput per minute recorded during the corresponding half-hour block. These estimated arrival rates for Terminals C and E are shown in Figure 6 for the period from 5:00 AM to 6:30 PM.

The service rate, μ, is harder to estimate. According to Massport in 2005, the maximum possible service rate, μ, of a single x-ray machine and metal detector was approximately two hundred customers per hour (3.3 customers per minute). However, the data provide only the throughput (TP), which underestimates the service *capacity* by not considering times where one or more servers was idle due to shorter queues. One remedy is to use the throughput during only those periods where the average waiting time stated in the data is nonzero, as the formation of queues indicates servers were likely working near maximum capacity. During these periods, the average throughput was closer to 2.8 customers per minute. Although this estimate varies greatly

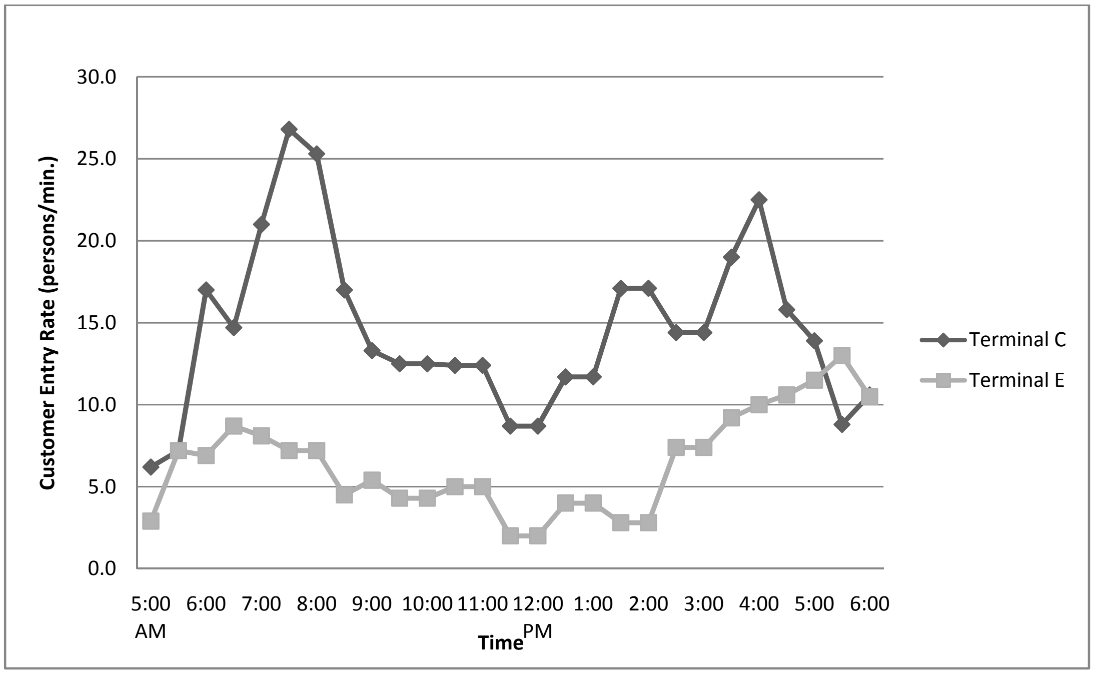

Figure 6: Customer arrival rates (in customers per minute) to Terminals C and E at Boston Logan International Airport on January 18, 2005, from 5:00 AM to 6:30 PM.

across time of day and terminals (for instance, if any of the machines were set to be more sensitive than usual to triggers, or if screeners were encouraged to work more quickly during peak hours to accommodate the additional load, service rates might be lower or higher than usual), it seems to be a reasonable estimate of the average service rate, based on calibration simulations.

Table 4 presents the results of our deterministic and dynamic server allocation heuristics applied to the Logan Airport arrival rate data. The columns labeled Deterministic Benchmark Allocation give the average waiting times (in minutes per customer) and the total number of servers switched under deterministic benchmark allocation applied to the deterministic fluid queue for each value of the switching time θ. As before, the number of switches is generally smaller under non-zero switching times than when switches can occur instantaneously, but there is a certain minimum number of switches (here, around eight), that must take place regardless of the lag time in order to accommodate variations in the customer arrival rates to the two queues. We will use the deterministic benchmark allocation as a baseline to assess the performance of the dynamic allocations for the stochastic scenarios.

	Deterministic Benchmark Allocation (Section 4.1)		Random Arrival Rates (Section 4.2)						Random Service Times (Section 4.3)				
			$\alpha = 0.1$, $\beta = 0.1$			$\alpha = 0.3$ $\beta = 0.3$							
			Wait (Min.)			Wait (Min.)			Wait (Min.)			Wait (Min.)	
θ	Wait (Min.)	No. of Switches	Bench-mark	Dyn-amic	% Impr.	Bench-mark.	Dyn-amic.	% Impr.	Bench-mark	Dynamic (Hybrid)	% Impr.	Dynamic (Neigh.)	% Impr.
0	1.89	18	2.05	1.94	5.4	5.90	4.03	31.7	2.36	**2.30**	2.5	**2.21**	6.4
5	2.09	8	2.27	2.25	0.9	4.86	4.45	8.4	2.49	**2.55**	-2.4	2.50	-0.4
10	2.30	8	2.46	2.42	1.6	5.06	4.66	7.9	2.81	**2.72**	3.2	**2.69**	4.3
15	2.49	9	2.64	2.59	1.9	5.30	4.85	8.5	2.94	2.90	1.4	**2.89**	1.7
30	3.05	9	3.18	3.21	-0.9	5.85	5.43	7.2	3.46	3.45	0.3	3.42	1.2

Table 4: Results of dynamic server allocation methods applied to the Logan Airport data, by switching time, θ (column 1). The columns labeled Deterministic Benchmark Allocation give the average waiting times (in minutes per customer) and the total number of servers switched under deterministic benchmark allocation applied to the deterministic fluid queue. The columns labeled Random Arrival Rates give the average waiting times for the deterministic benchmark allocation ("Benchmark") and the dynamic allocation ("Dynamic") applied to the randomized arrival rate fluid queues with ($\alpha = 0.1$, $\beta = 0.1$) and ($\alpha = 0.3$, $\beta = 0.3$), and the percentage improvement of the dynamic allocation over the deterministic benchmark allocation. The columns labeled Random Service Times give average waiting times for the deterministic benchmark allocation ("Benchmark"), the hybrid dynamic allocation ("Hybrid") and the nearest neighbor dynamic allocation ("Neigh.") applied to the randomized service time fluid queues, as well as the percentage improvement the dynamic hybrid and nearest neighbor allocations have over the deterministic benchmark allocation. Numbers in boldface indicate a statistically significant difference from the benchmark deterministic allocation at a 95% confidence level.

The columns labeled Random Arrival Rates in Table 4 show the average waiting time in the system with randomized arrival rates under the benchmark and dynamic allocations, for the cases of ($\alpha = 0.1$, $\beta = 0.1$) and ($\alpha = 0.3$, $\beta = 0.3$). As we observed for the synthetic data, in a system with a high level of uncertainty about the actual arrival rates ($\alpha = 0.3$, $\beta = 0.3$), dynamic allocation yields a large reduction (at least 7%) in the average waiting time over the original deterministic benchmark allocation. So even when using arrival rate data from a real rather than ideal scenario, we see that dynamically allocating the servers using dynamic programming methods can significantly reduce the average waiting time of customers if there is a great deal of uncertainty in the arrival rates.

The columns of Table 4 labeled Random Service Times compare the average waiting times per customer over 500 simulation trials for the hybrid and nearest neighbor dynamic allocations to those for the deterministic benchmark allocation, for the case of random service times. As we observed for the synthetic data, although the dynamic allocation heuristics offer in some cases a statistically significant reduction in the average waiting time, the magnitude of these reductions is small. Dynamic allocation does not appear to be necessary when the only randomness in the system occurs with respect to individuals rather than at an aggregate level. It is possible that systematic changes to service times spanning one or more decision epochs (e.g. such as might be caused by equipment failure or an unexpected change in security protocol) might have a greater influence on the optimal server allocation.

We have demonstrated by simulation (though not in practice on a real queue) the effectiveness of dynamically allocating servers in response to the stochastically evolving system state using real queue arrival rate data. In the next section, we examine whether the decisions of when to dynamically switch servers between queues align with our intuition.

6.3 General observations

Because we would like queue managers to be able to decide quickly whether to order a switch of servers between queues, it would be useful to distill the optimal switching policy into simple heuristics based on system state characteristics. For instance, based on our intuition, we might expect the following general rules to hold:

1. Switches should be initiated less frequently for larger values of the switching time, θ.

2. The policy should be exhaustive: no servers should be switched from Queue A to Queue B until Queue A has been depleted.

3. The policy should never switch servers from a longer queue to a shorter queue.

Indeed, each of these three properties was proven to hold in simpler server allocation frameworks as described earlier in Section 3. However, *none* hold in our framework involving nonhomogeneous arrivals and non-zero switching times. For instance, as was shown in Tables 3 and 4 discussed earlier, the number of switches conducted does not always decrease as θ, the switching time, increases. When switches require a long period of time, additional servers may need to be switched to accommodate queues that grow during that time. It was also observed in the simulation results that the DP's decision may occasionally be to switch servers *away* from a longer queue, to a shorter queue. This is in contradiction with many results found in the literature (see, for instance, Liu, Nain and Towsley [14]) for systems with or without switching times, and is most likely due to the fact that the decisions made in our models are allowed to anticipate future changes in the arrival rates at each queue. Although Queue A might be longer than Queue B in the current period, knowledge that Queue B's arrival rate might suddenly increase in the near future could support a decision to move servers *away* from Queue A.

Given the ineffectiveness of simple heuristics, two more sophisticated heuristics were tested: one based on a statistical analysis of the number of servers switched as a function of system state and knowledge of future arrival rates, and another based on a heuristic by Duenyas and Van Oyen [7] that adapts the $c\mu$-rule to accommodate non-zero switching times. However, neither method performed better than the deterministic benchmark allocation, and they often increased queuing delays.

There is no known simple relationship between the observed state of the system at time t and the optimal switching decision to be made at time t because of the complicated ramifications such a decision will have throughout the time horizon. Thus, relying on intuition or simple heuristics to determine when servers should be switched from one queue to another is not an effective queue control strategy. This is an important finding because managers who are already dynamically allocating servers between queues based on their intuition (as was observed at San Francisco International Airport) may unknowingly be *increasing* queuing delays.

7 Conclusions

We have studied a queue control problem of dynamically allocating servers between parallel queues in a system having nonhomogeneous arrivals and where switches do not occur instantaneously. Intuitive queue control rules that work well in systems with homogeneous arrivals and instantaneous switches do not work well in this framework. Specifically, the number of switches need not decrease as the switching time increases, optimal policies need not be exhaustive, and servers can indeed be switched to queues having shorter queue lengths. Because of this, simple heuristics to determine when switches should take place are ineffective. We have used approximate dynamic programming techniques within a fluid queuing framework to demonstrate that dynamically allocating servers between queues has the potential to significantly reduce waiting times. In particular, we find that dynamic allocation of servers is most effective when the uncertainty in customer arrival rates is greatest. An important extension to this work would be to consider switches between more than two queues. While this could provide additional flexibility to a dynamic allocation, it would also increase dramatically the complexity of the problem.

8 Acknowledgments

The author would like to thank the Alfred P. Sloan Foundation who funded this work, Massport for providing data, Arnold Barnett for his guidance with an earlier version of this work, and the helpful comments of three anonymous reviewers.

References

[1] Baras, J. S., Dorsey, A. J., and Makowski, A. M. (1985) Two competing queues with linear costs: The μc rule is often optimal. *Advances in Applied Probability*, **17**, 186–209.

[2] Bell, S. L. and Williams, R. J. (2001) Dynamic scheduling of a system with two parallel servers in heavy traffic with resource pooling: Asymptotic optimality of a threshold policy. *The Annals of Applied Probability*, **11**(3), 608–649.

[3] Berman, O. and Larson, R. C. (2004) A queueing control model for retail services having back room operations and cross-trained workers. *Computers and Operations Research*, **31**(2), 201–222.

[4] Bertsekas, D. P. (2000) *Dynamic Programming and Optimal Control*, volume 1, second edition, Athena Scientific, Belmont, Massachusetts.

[5] Buyokkoc, C., Varaiya, P., and Walrand, J. (1985) The $c\mu$ rule revisited. *Advances in Applied Probability*, **17**, 237–238.

[6] Chen, H. and Yao, D. D. (1993) Dynamic scheduling of a multiclass fluid network. *Operations Research*, **41**(6), 1104–1115.

[7] Duenyas, I. and Van Oyen, M. P. (1996) Heuristic scheduling of parallel heterogeneous queues with set-ups. *Management Science*, **42**(6), 814–829.

[8] Fisher, M., Kubicek, C., McKee, P., Mitrani, I., Palmer, J., and Smith, R. (2004) Dynamic allocation of servers in a grid hosting environment, in *Proceedings of the fifth IEEE/ACM International Workshop on Grid Computing.*

[9] Harrison, J. M. (1996) The BIGSTEP approach to flow management in stochastic processing networks, in F. P. Kelly, S. Z. and Ziedins, I. (eds.) *Stochastic Networks: Theory and Applications*, Oxford University Press, pp. 57–90.

[10] Harrison, J. M. and López, M. J. (1999) Heavy traffic resource pooling in parallel-server systems. *Queueing Systems*, **33**, 339–368.

[11] Hofri, M. and Ross, K. W. (1987) On the optimal control of two queues with server setup times and its analysis. *SIAM Journal of Computing*, **16**, 399–420.

[12] Koole, G. (1997) Assigning a single server to inhomogeneous queues with switching costs. *Theoretical Computer Science*, **182**, 203–216.

[13] Koole, G. (1998) Structural results for the control of queueing systems using event-based dynamic programming. *Queueing Systems*, **30**, 323–339.

[14] Liu, Z., Nain, P., and Towsley, D. (1992) On optimal polling policies. *Queueing Systems*, **11**, 59–83.

[15] Lu, F. V. and Serfozo, R. F. (1984) M/M/1 Queueing decision processes with monotone hysteretic policies. *Operations Research*, **32**, 1116–1132.

[16] Mandelbaum, A., Massey, W. A., Reiman, M. I., Rider, B., and Stolyar, A. L. (2002) Queue lengths and waiting times for multiserver queues with abandonment and retrials. *Telecommunications Systems*, **21**, 149 – 171.

[17] Martonosi, S. E. (2005) *An operations research approach to aviation security*, Ph.D. Thesis, Massachusetts Institute of Technology.

[18] Mason, A. J., Ryan, D. M., and Panton, D. M. (1998) Integrated simulation, heuristic and optimisation approaches to staff scheduling. *Operations Research*, **46**(2), 161–175.

[19] Massport, Airport statistics, accessed on December 3, 2010 from http://www.massport.com/logan-airport/about-logan/Pages/LoganStatistics.aspx.

[20] Moder, J. J. and Phillips, Jr., C. R. (1962) Queuing with fixed and variable channels. *Operations Research*, **10**, 218–31.

[21] Newell, G. F. (1971) *Applications of Queueing Theory*, 1st edition, Chapman and Hall, London.

[22] Palmer, J. and Mitrani, I. (2004) Optimal server allocation in reconfigurable clusters with multiple job types, in *Proceedings of the 2004 International Conference on Computational Science and its Applications, Part II*, LNCS 3044, pp. 76–86.

[23] Palmer, J. and Mitrani, I. (2005) Optimal and heuristic policies for dynamic server allocation. *Journal of Parallel Distributed Computing*, **65**, 1204–1211.

[24] Panayiotou, C., Fu, M., and Howell, W. C. (2005) Online traffic light control through gradient estimation using stochastic fluid models, in *Proceedings of the IFAC 16th Triennial World Congress*, Prague.

[25] Powell, W. B. (1996) A stochastic formulation of the dynamic assignment problem, with an application to truckload motor carriers. *Transportation Science*, **30**(3), 195–219.

[26] Ridley, A. D., Fu, M. C., and Massey, W. A. (2003) Fluid approximations for a priority call center with time-varying arrivals, in Chick, S., Sánchez, P. J., Ferrin, D., and Morrice, D. J. (eds.) *Proceedings of the 2003 Winter Simulation Conference.*

[27] Rosa-Hatko, W. and Gunn, E. A. (1997) Queues with switchover - a review and critique. *Annals of Operations Research*, **69**, 299–322.

[28] Sethuraman, J. and Squillante, M. S. (1999) Optimal stochastic scheduling in multiclass parallel queues, in *Proceedings of ACM SIGMETRICS '99*, Atlanta, GA.

[29] Squillante, M. S., Xia, C. H., Yao, D. D., and Zhang, L. (2001) Threshold-based priority policies for parallel-server systems with affinity scheduling, in *Proceedings of the American Control Conference*, Arlington, VA.

[30] Vandergraft, J. S. (1983) A fluid flow model of networks of queues. *Management Science*, **29**(10), 1198–1208.

[31] Whitt, W. (1999) Dynamic staffing in a telephone call center aiming to immediately answer all calls. *Operations Research Letters*, **24**, 205–212.