Numerical Methods in Optimization and Performance Analysis of Queueing Networks

by

Yiqun Song, B.Sc.

A thesis submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

**Master of Science
in
Probability and Statistics**

School of Mathematics and Statistics
Ottawa-Carleton Institute of Mathematics and Statistics
Carleton University
Ottawa, Ontario

©2014

Yiqun Song

To my husband, Zheng, who knows me better than anyone else does in the world.

Abstract

We consider control problems in many-server queueing networks, with impatient customers. In particular, we focus on models with two customer classes and one, or two, service stations. Each customer can receive service only once, possibly in either of the available stations; and can also abandon while waiting in queue. The service is assumed exponential with rates depending on station type only.

Our objective is to find a policy minimizing the expected discounted linear holding cost of customers in the queue. The problem is considered in the so-called Quality and Efficiency-Driven regime, when both the arrival rates and the number of servers are large, and the system is critically loaded. We use Markov Decision Processes techniques to investigate properties of an optimal policy, as well as to plot the approximated optimal cost function for sets of given parameters. Our conclusions are: (a) an optimal policy depends on the total number of customers in system only; (b) it is optimal to keep idleness in the station with lower service rate; (c) an optimal policy is “bang-bang”, dictating to rearrange the customers in such a way that, at any given moment of time, only one type of customers is waiting (and the customers of the other type are all being served). The decision, we argue, depends on whether the derivative of the optimal cost function is greater than the ratio $\frac{c_1 - c_2}{\theta_1 - \theta_2}$; with c_i and θ_i being, respectively, the holding cost rate and the abandonment rate of class i .

The second part of the thesis introduces a simple algorithm for simulating G/G/N queueing system, which is then extended to more complex networks.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Dr. Gennady Shaikhet, for all the guidance, support, patience and encouragement he provided throughout my graduate study. I greatly appreciate the freedom he gave me to choose my thesis topic, which made my research experience a truly enjoyable journey. I am deeply thankful for his devoted time and effort in making this thesis possible. I am particularly grateful for his generous financial help that saw me through to the end.

I wish to express special thanks to Dr. Matthew Kennedy for his understanding and encouragement during a very difficult time of my thesis work. I especially value our friendship.

I must thank all the lovely people I met in the Graduate Student Lab for their kindness and warm friendships. Their company made my study life less lonely, more colorful, and more memorable.

Finally, I wish to thank my family. Thanks to my parents for their love, devotion, unconditional support and unwavering confidence in me. Thanks to my husband for his love, patience, tolerance, unfailing help, continuous encouragement and always being there for me.

Table of Contents

Abstract	iii
Acknowledgments	iv
Table of Contents	v
List of Figures	vii
Notations	viii
1 Introduction	1
2 Asymptotic Analysis and Control of Many-Server Queueing Systems	6
2.1 The QED regime for M/M/N queues	6
2.2 The QED regime for multi-class many-server queueing systems	8
2.2.1 The fluid model	8
2.2.2 Important estimates	10
2.3 Control of multi-class many-server queues	12
2.3.1 Original model	12
2.3.2 Diffusion model	13
2.3.3 Dimension reduction of pool-dependent models	15
3 Markov Decision Process	18
3.1 Introduction	18
3.1.1 Preliminaries	18
3.1.2 Value iteration algorithm	19
3.1.3 Policy iteration algorithm	21

3.2	Optimal control of V model	22
3.2.1	Markov decision problem	23
3.2.2	Numerical analysis	25
3.2.3	Proof	30
3.3	Optimal control of N model	32
3.3.1	Markov decision problem	33
3.3.2	Numerical analysis	35
3.3.3	Proof	39
3.4	Optimal control of X model	42
3.4.1	Markov decision problem	43
3.4.2	Numerical analysis	44
3.4.3	Proof	47
4	Discrete-Event Simulation	51
4.1	Introduction	51
4.2	Simulation of G/G/1 model	52
4.3	Simulation of G/G/N model	54
4.4	Three extended examples	56
4.4.1	Simulation of G/G/N with abandonment	56
4.4.2	Simulation of V model	59
4.4.3	Simulation of N model	62
	Bibliography	65
	Appendix A Code for Markov Decision Process	68
A.1	Optimal control of V model	68
A.2	Optimal control of N model	75
A.3	Optimal control of X model	83
	Appendix B Code for Discrete-Event Simulation	93
B.1	Simulation of G/G/N model	93
B.2	Simulation of G/G/N with abandonment	95
B.3	Simulation of V model	98

List of Figures

1.1	Some canonical queueing models	2
2.1	A system with two customer classes and three server pools	8
2.2	A non-critically loaded system	9
2.3	A critically loaded system	10
3.1	V model	22
3.2	Optimal policy for V model: $c_1 = 11$	26
3.3	Optimal policy for V model: $c_1 = 7$	27
3.4	Optimal policy for V model: $c_1 = 14$	27
3.5	Optimal policy for V model: $c_1 = 16$	28
3.6	V model: relation between $\frac{dV_\alpha^*(x_1, x_2)}{d(x_1 + x_2)}$ and $\frac{c_1 - c_2}{\theta_1 - \theta_2}$	30
3.7	N model	33
3.8	Optimal policy for N model: $c_1 = 6$	36
3.9	Optimal policy for N model: $c_1 = 8$	36
3.10	Optimal policy for N model: $c_1 = 9$	37
3.11	Optimal policy for N model: $c_1 = 12$	37
3.12	N model: relation between $\frac{dV_\alpha^*(x_1, x_2)}{d(x_1 + x_2)}$ and $\frac{c_1 - c_2}{\theta_1 - \theta_2}$	38
3.13	X model	42
3.14	Optimal policy for X model	46
3.15	X model: relation between $\frac{dV_\alpha^*(x_1, x_2)}{d(x_1 + x_2)}$ and $\frac{c_1 - c_2}{\theta_1 - \theta_2}$	47
4.1	G/G/1 model	52
4.2	G/G/N model	54
4.3	G/G/N with abandonment	57
4.4	V model	59
4.5	N model	62

Notations

e	$=(1, 1, \dots, 1)$. The dimension of vector e may vary from one another.
e_i	The i th coordinate vector.
$v \cdot u$	A scalar product of two vectors v and u of the same dimension.
x_e	$=e \cdot x = x_1 + \dots + x_I$ for $x \in \mathbb{R}^I$.
$x \succeq y$	$x \geq y$ and $x(i) > y(i)$ for at least one i for $x, y \in \mathbb{R}^I$.
$x \wedge y$	$=\min\{x, y\}$.
$x \vee y$	$=\max\{x, y\}$.
x^+	$=\max\{x, 0\}$.
x^-	$=\max\{-x, 0\}$.
$\ x\ $	$=\sum_{i=1}^I x_i $. A norm of $x \in \mathbb{R}^I$.
$\ X\ _t^*$	$=\sup_{0 \leq u \leq t} \ X(u)\ $. A norm of an $\mathbb{R}^I$ -valued function.
$\mathbb{Z}_+$	Set of positive integers.
$\mathbb{P}_x^\pi$	Probability measure determined by policy π and initial state x .
$\mathbb{E}_x^\pi$	Expectation operator with respect to $\mathbb{P}_x^\pi$.
$\mathbb{1}_A$	Indicator function. If A is true, $\mathbb{1}_A = 1$; otherwise, $\mathbb{1}_A = 0$.
$\mathcal{I}$	$=\{1, 2, \dots, I\}$, the index set for customer classes.
$\mathcal{J}$	$=\{1, 2, \dots, J\}$, the index set for server pools.
$X_i^n(t)$	The total number of class- i customers in the system at time t .
$Y_i^n(t)$	The number of class- i customers in queue at time t .
$Z_j^n(t)$	The number of idle servers in pool j at time t .
$\Psi_{ij}^n(t)$	The number of class- i customers served in pool j at time t .
$\hat{X}^n(t)$	X^n in diffusion scaling.

w	$= (w_1, w_2)$, the fraction of total queued customers in class 1 and 2.
v	$= (v_1, v_2)$, the fraction of total idle servers in pool 1 and 2.
S	State space.
(x_1, x_2)	A state described by the numbers of customers in class 1 and 2.
$A(x_1, x_2)$	Set of actions available at state (x_1, x_2) .
f	A stationary policy.
f^*	Optimal policy for certain performance criterion.
$\mathcal{F}$	Set of all stationary policies.
α	The discount factor.
$V_\alpha(x_1, x_2, f)$	Expected discount cost under policy f given initial state (x_1, x_2) .
$V_\alpha^*(x_1, x_2)$	$= \min_{f \in \mathcal{F}} V_\alpha(x_1, x_2, f)$, the optimal discount cost.

Chapter 1

Introduction

Since introduced by Erlang in 1920s the queueing theory has become a fundamental research area with a wide range of practical applications. Various queueing models have been developed for analyzing real-life problems. Such models are generally classified into two types: descriptive and prescriptive [1]. Descriptive models are used to analyze actual situations, and play a dominant role in a standard queueing theory. Prescriptive models, in turn, prescribe the desired behavior, and are generally referred to as *design and control of queues*. They serve to prescribe optimal system parameters (such as optimal service rate, the optimal number of servers), as well as an optimal service discipline. Design models are generally static in nature; that is, cost or profit functions are superimposed and to be optimized with respect to parameters like arrival rate λ , service rate μ , or the number of servers N [1]. In contrast, control models are usually more dynamic in nature and focus on determining the optimal policy.

Unlike the passive development of design models, research in control models has been growing at an ever-increasing rate over the past decades. Control models have been applied to solve numerous problems in various fields, including call center systems [2], energy management systems [3], health care systems from kidney allocation [4] to emergency medical services [5], computers and communication systems [6], and manufacturing systems [7].

Certain issues have been discussed in the control of queueing models. Routing and scheduling control (RSC) problem is among the most interesting and challenging problems. The routing problem is *to decide, upon a customer's arrival, to which of the idle servers, if any, should the customer be routed*. The scheduling problem is *to determine, upon a service completion, which of waiting customers, if any, should*

be scheduled to get service. Complex model structure often leads to various choices of routing and scheduling policy. Garnett and Mandelbaum [8] introduced some canonical queueing models, as shown in Figure 1.1.

Figure 1.1: Some canonical queueing models

For example, in a V model, a single pool of servers handles two (or more) classes of customers. In a N model, pool 1 serves customers of class 1, while pool 2 serves customers of both classes. In a X model, two classes of customers can be served by either of the two pools. All the models in Figure 1.1 can be enriched by allowing abandonment of waiting customers.

Besides model structure, aspects such as the number of servers, heavy traffic regime and cost function also contribute to the diversity of queueing control problem. For instance, Van Mieghem [9] and Mandelbaum and Stolyar [10] considered a queueing model with I classes of customers and J non-identical servers working in parallel with convex cost function in the conventional, *single-server*, heavy traffic regime. They obtained the generalized $c\mu$ -rule ($Gc\mu$) as the asymptotically optimal scheduling policy. The policy can be described as follows: at each time t , when server j becomes free, the next customer scheduled to it is a class- i customer such that $i \in \operatorname{argmax}_i C'_i(Q_i(t))\mu_{ij}$. Here C' is the derivative of cost function C , $Q_i(t)$ is the queue length of class- i customers at time t , and μ_{ij} is the average service rate of class- i customers served by server j .

An alternative to index control is threshold control. Bell and Williams [11] looked at a two-parallel-server system structured as a N model under linear holding cost. They considered a parameter regime in which the system is heavily loaded and allows complete resource pooling. A threshold policy is proved to be asymptotically optimal; that is, the priority of service is determined by whether or not the queue length has exceeded the threshold. The results were extended to general parallel server systems in [12].

In many-server queueing systems, the RSC problems turn out to be more difficult than in parallel single-server systems. An asymptotic approach, where a diffusion control problem is obtained in the heavy traffic limit through parametrization of the model, is often used to simplify the problem. The works of Harrison and Zeevi [13] and Atar, Mandelbaum and Reiman [14] analyzed V model in which a pool of statistically identical servers caters to multiple classes of impatient servers. Harrison and Zeevi [13] considered a Markovian model in the Quality- and Efficiency-driven (QED) regime and found the optimal scheduling showing an extremal or “bang-bang” character. Atar, Mandelbaum and Reiman [14] established a relation between their scheduling problem and a diffusion control problem in the QED regime. Then, by solving the diffusion control problem via Hamilton-Jacobi-Bellman (HJB) equation, they attained the asymptotically optimal policy for the original queueing system.

Tezcan and Dai [15] studied a many-server N model under some conditions on service, abandonment and cost parameters. To be specific, they assumed the service rates are pool-dependent and customers of the cheaper class are more likely to abandon the system. In QED regime, a $c\mu$ -type static priority policy is proved asymptotically optimal for this queueing system. One needs only the holding cost and service rate to establish the policy.

For many-server models with I customer classes and J server pools, Gurvich and Whitt [16] proposed a control policy named Fixed-Queue-Ratio (FQR) rule. It indicates that, when a service ends, the next customer scheduled should be the longest-waiting one of the class which is eligible to be served and has the queue length most exceeding a specified proportion of the total queue length. This results in a dimension reduction; that is, the vector-valued queue-length process is asymptotically evolving as a one-dimensional process. Atar, Mandelbaum and Shaikhet [17] dealt with the similar queueing model in two special cases. First, when the service rates depend only

on the pool, the control problem is shown to be asymptotically reducible to a one-dimensional diffusion control problem. Secondly, they present certain simplifications for the case of the class-dependent service rates.

However, a very interesting case of many-server models with pool-dependent service rates has not been solved explicitly yet. We consider, for example, a two-class V model in which the holding cost rates c_1 , c_2 and the abandonment rates θ_1 , θ_2 satisfy

$$c_1 > c_2 \quad \text{and} \quad \theta_1 > \theta_2. \quad (1.1)$$

This condition indicates that, in order to minimize the total linear holding cost, one must make a trade-off between low cost rate and high abandonment rate.

Shaikhet [18] suggested the optimal policy of this model is “bang-bang”. Based on the results of [17], this two-dimensional control problem can be reduced to a one-dimensional one in which the only control variable is the allocation of waiting population between the classes. Then, the corresponding one-dimensional HJB equation shows that the control variable can only take two extreme values. It implies that the optimal control might be a threshold policy.

In this thesis, we find the optimal control and provide a detailed proof by formulating the problem as a Markov Decision Process (MDP). The results confirm the conjecture in [18]; that is, the optimal scheduling is proved to be a threshold policy where the priority of customer class is decided based on the total number of customers in the system. By applying the same approach, we also solve the optimal control problems for pool-dependent N and X models, for which the cost and abandonment rates satisfy (1.1). The results are similar to those of the above V model.

The other part of the thesis presents an approach of discrete-event simulation. Simulation is an effective approach to analyze complex system. In queueing theory, simulation is often used to estimate the measures of performance of a queueing system in steady state. As a result, one can compare different control policies with the estimates. For example, Garnett and Mandelbaum [8] used simulation to demonstrate that various control policies end in dramatic differences in the performance of system. In [19], Armony and Ward provided simulation results to show that their threshold routing policy outperforms the routing policies commonly used in call centers such as faster-server-first (FSF) and longest-idle-server-first (LISF) policies. Similar usage of simulation can also be found in [20] and [21].

In order to describe the contents of the thesis, we introduce a shorthand notation

of queueing system. $A/B/N$ denotes a N -server queueing system, where A and B indicate the interarrival time distribution and the service time distribution, respectively. Some standard symbols for A and B are M (exponential), D (deterministic) and G (general). The symbol G represents a general probability distribution; that is, no assumption is made about the precise form of the distribution. Thus, for example, $G/M/2$ means a two-server system with general distributed interarrival times and exponentially distributed service times.

It is well known that system $M/M/N$ can be easily simulated by using the memoryless property of exponential distribution. Meanwhile, the algorithm of simulating $G/G/1$ has been provided by Ross [22]. However, simulation of the most general system $G/G/N$ remains a challenging problem. In this thesis, we present a simple algorithm for $G/G/N$. The simulation approach can be applied to more complex queueing systems, for example, any of the models in Figure 1.1. We include three examples in the thesis: $G/G/N$ with abandonment, V model and N model.

Chapter 2

Asymptotic Analysis and Control of Many-Server Queueing Systems

In this chapter, we will review the Quality and Efficiency Driven (QED) regime, which is characterized by both the arrival rates and the number of servers being very large, with the utilization close to 1. It will be shown that, under such a regime, the system fluctuates around the so-called *static fluid model* with fluctuations being of much smaller order than the arrival rates. We characterize the fluctuations in the case of single-class single-station network and then present the method of controlling fluctuations for more general network structures.

2.1 The QED regime for M/M/N queues

The QED regime was first formalized by Halfin and Whitt [23] for the $M/M/N$ model. A striking feature of the regime is that, despite the critical load, the delay probability (i.e., the probability that an arriving customer would have to wait) does not converge to 1, but to some constant strictly less than 1. That is to say, the system achieves high levels of both server efficiency and service quality. In what follows we elaborate in more detail.

Consider a sequence of $M/M/N$ systems indexed by $n \rightarrow \infty$. Assume that the number of servers $N = n$. Let λ^n and μ^n be the arrival and the service rates, satisfying $\frac{\lambda^n}{n} \rightarrow \mu$ and $\mu^n \rightarrow \mu$ for some constant $0 < \mu < \infty$. Define the system utilization $\rho^n = \frac{\lambda^n}{n\mu^n}$ and assume

$$\sqrt{n}(1 - \rho^n) \rightarrow \beta, \quad \text{for some } \beta \in (0, \infty). \quad (2.1)$$

Define the centered, normalized queue length as

$$\hat{X}^n(t) = \frac{X^n(t) - n}{\sqrt{n}}, \quad t \in [0, \infty). \quad (2.2)$$

The following theorem states that, with such parameters, the system is almost full, fluctuating around the *static average* value of n , with fluctuations being of the order $O(\sqrt{n})$.

Theorem 2.1. (*Halfin and Whitt, 1981 [23]*).

1. Assume $\hat{X}^n(0) \Rightarrow \hat{X}(0)$. Then $\hat{X}^n$ converges weakly to a diffusion process $\hat{X}$ characterized by

$$\hat{X}(t) = \hat{X}(0) + \int_0^t b(\hat{X}(s))ds + \sqrt{2\mu}W(t).$$

Here $W(t)$ is a standard Brownian motion, and the drift b is given as

$$b(x) = \begin{cases} -\mu\beta & x \geq 0, \\ -\mu(x + \beta) & x < 0. \end{cases}$$

2. The probability of delay has a nondegenerate limit as follows:

$$\lim_{N \rightarrow \infty} P(X^N(\infty) \geq N) = \alpha, \quad 0 < \alpha < 1, \quad (2.3)$$

where α is related to β from (2.1) via $\alpha = [1 + \beta\Phi(\beta)/\phi(\beta)]^{-1}$. (Here Φ and ϕ are, respectively, the distribution and density functions of the standard normal distribution).

The results were later extended for M/M/N queues with abandonment by Garnett, Mandelbaum and Reiman [24]. Because of its desirable features, the QED regime has also been considered for multi-class many-server systems such as modern telephone call center (see [2] for a review).

2.2 The QED regime for multi-class many-server queueing systems

We study a Markovian queueing system with I customer classes and J server pools (see Figure 2.1 as an example). Let $\mathcal{I} = \{1, \dots, I\}$ and $\mathcal{J} = \{I + 1, \dots, I + J\}$ denote the index sets for customer classes and server pools, respectively.

Consider a sequence of the above systems indexed by $n \rightarrow \infty$. For $i \in \mathcal{I}$, customers of class i arrive according to a Poisson process of rate λ_i^n , and their patience time is exponentially distributed with rate θ_i^n . A server pool j , for each $j \in \mathcal{J}$, consists of N_j^n statistically identical and independent servers, providing an exponential (μ_{ij}^n) service to class- i customers.

Figure 2.1: A system with two customer classes and three server pools

2.2.1 The fluid model

In the QED heavy traffic regime, the system is supposed to fluctuate around the so-called *critically loaded static fluid model*, which was proposed by Harrison and López [25]. For this reason, we assume that there exist constants $\{\lambda_i\}$, $\{\mu_{ij}\}$, $\{\nu_j\}$, $\{\theta_i\}$ so that the system's parameters satisfy

$$\lambda_i^n = n\lambda_i + O(\sqrt{n}), \quad i \in \mathcal{I} \quad (2.4)$$

$$n\mu_{ij}^n = n\mu_{ij} + O(\sqrt{n}), \quad i \in \mathcal{I}, \quad j \in \mathcal{J} \quad (2.5)$$

$$n\theta_i^n = n\theta_i + O(\sqrt{n}), \quad i \in \mathcal{I}, \quad (2.6)$$

$$N_j^n = n\nu_j + O(\sqrt{n}), \quad j \in \mathcal{J} \quad (2.7)$$

These fluid level parameters $\{\lambda_i\}$, $\{\mu_{ij}\}$, $\{\nu_j\}$ should guarantee that the system is critically loaded in a following sense: there exists a unique optimal solution (ξ^*, ρ^*) to the below linear program, satisfying $\sum_{i \in \mathcal{I}} \xi_{ij}^* = 1$ for all $j \in \mathcal{J}$ (and consequently $\rho^* = 1$):

$$\text{Minimize } \rho \in \mathbb{R}_+ \text{ subject to } \begin{cases} \sum_{j \in \mathcal{J}} \nu_j \mu_{ij} \xi_{ij} = \lambda_i, \\ \sum_{i \in \mathcal{I}} \xi_{ij} \leq \rho, & i \in \mathcal{I}, j \in \mathcal{J} \\ \xi_{ij} \geq 0. \end{cases} \quad (2.8)$$

One gets a deterministic *allocation* matrix ξ^* , where for $(i, j) \in \mathcal{I} \times \mathcal{J}$, the entry ξ_{ij}^* represents the fraction of pool- j work dedicated to class- i customers on the fluid level.

For example, consider a fluid system with parameters $n = 100$, $\lambda_1 = 2$, $\lambda_2 = 1$, $\mu_{11} = 2$, $\mu_{12} = 2$, $\mu_{21} = 2$, $\mu_{22} = 1$, $\nu_1 = 1$, and $\nu_2 = 1$. If we allocate the fluid as $\xi_{11} = 1$, $\xi_{12} = 0$, $\xi_{21} = 0$, and $\xi_{22} = 1$, the utilization of both pools are $\sum_i \xi_{i1} = \sum_i \xi_{i2} = 100\%$. But the system is not critically loaded because 100% is not the optimal ρ . It can be shown that the utilization decreases to 75% if we reallocate the fluid as $\xi_{11} = 0.25$, $\xi_{12} = 0.75$, $\xi_{21} = 0.5$, and $\xi_{22} = 0$.

Figure 2.2: A non-critically loaded system

An example of critically loaded system is that $n = 100$, $\lambda_1 = 7.5$, $\lambda_2 = 2$, $\mu_{11} = 4$, $\mu_{12} = 7$, $\mu_{21} = 2$, $\mu_{22} = 4$, $\nu_1 = 1$, and $\nu_2 = 1$. In this case, $\xi_{11}^* = 1$, $\xi_{12}^* = 0.5$, $\xi_{21}^* = 0$, and $\xi_{22}^* = 0.5$ is the unique optimal allocation with $\rho^* = 1$.

Figure 2.3: A critically loaded system

Throughout the thesis, we keep the notation ξ_{ij}^* . We also denote

$$a1\psi_{ij}^* = \xi_{ij}^* \nu_j, \quad x_i^* = \sum_{j \in \mathcal{J}} \xi_{ij}^* \nu_j, \quad i \in \mathcal{I}, \quad j \in \mathcal{J}. \quad (2.9)$$

2.2.2 Important estimates

Our models of interest are the so-called V , N and X models that introduced in Figure 1.1. Let $X_i^n(t)$ denote the total number of class- i customers in the system at time t . Considering work-conservation (i.e. a server is allowed to be idle only when no customer that it can serve is waiting), it has been shown in Atar [26] that in V and N models, where the graph of the activities (i, j) in the *static fluid model* is a tree, the quantities

$$\hat{X}_i^n(t) = \frac{X_i^n(t) - nx_i^*}{\sqrt{n}}, \quad i \in \mathcal{I} \quad (2.10)$$

satisfy, for each $\gamma > 0$

$$\mathbb{P}(\|\hat{X}^n\|_t^* > n^\gamma) \rightarrow 0 \quad \text{as} \quad n \rightarrow \infty. \quad (2.11)$$

In other words, the system fluctuates around the static fluid model.

A model is called pool-dependent if the service rates depend only on the pool ($\mu_{ij}^n = \mu_j^n$). If a X model is in the pool-dependent case, a weaker version of (2.11) was provided in [17]:

$$\mathbb{P}(|\hat{X}_e|_t^* > n^\gamma) \rightarrow 0 \quad \text{as } n \rightarrow \infty. \quad (2.12)$$

Based on the above claims, we present some important estimates needed to establish Markov decision processes in Chapter 3.

V model

In a two-class V model, $\mathcal{I} = \{1, 2\}$ and $\mathcal{J} = \{1\}$. $X_1^n(t)$ and $X_2^n(t)$ fluctuate within the rectangle region

$$\{(X_1^n(t), X_2^n(t)) | X_i^n(t) \in [nx_i^* - n^{\frac{1}{2}+\gamma}, nx_i^* + n^{\frac{1}{2}+\gamma}], \quad \forall i \in \mathcal{I}\}. \quad (2.13)$$

In addition, $N^n + O((N^n)^{\frac{2}{3}})$ is an upper bound of the total number of customers for the V model in the QED limit; that is,

$$\mathbb{P}(X_1^n(t) + X_2^n(t) \geq N^n + O((N^n)^{\frac{2}{3}})) \rightarrow 0 \quad n \rightarrow \infty, \quad (2.14)$$

N model

In a N model, $\mathcal{I} = \{1, 2\}$ and $\mathcal{J} = \{1, 2\}$. Assume that pool 1 serves customers of class 1 and pool 2 serves customers of both classes. Same as above, $X_1^n(t)$ and $X_2^n(t)$ fluctuate within the rectangle region

$$\{(X_1^n(t), X_2^n(t)) | X_i^n(t) \in [nx_i^* - n^{\frac{1}{2}+\gamma}, nx_i^* + n^{\frac{1}{2}+\gamma}], \quad \forall i \in \mathcal{I}\}; \quad (2.15)$$

The asymptotically upper bounds of the number of customers in class 1 and class 2 are as below.

$$\mathbb{P}(X_1^n(t) \geq N_1^n + N_2^n) \rightarrow 0 \quad n \rightarrow \infty; \quad (2.16)$$

$$\mathbb{P}(X_2^n(t) \geq N_2^n) \rightarrow 0 \quad n \rightarrow \infty; \quad (2.17)$$

X model

In a X model, $\mathcal{I} = \{1, 2\}$ and $\mathcal{J} = \{1, 2\}$. Customers of both classes can be served by either of the two pools. According to (2.12),

$$\mathbb{P}(X_1^n(t) + X_2^n(t) \geq N_1^n + N_2^n + O((N_1^n + N_2^n)^{\frac{2}{3}})) \rightarrow 0 \quad n \rightarrow \infty. \quad (2.18)$$

Hence, $N_1^n + N_2^n + O((N_1^n + N_2^n)^{\frac{2}{3}})$ is an upper bound of the total number of customers in the QED limit.

2.3 Control of multi-class many-server queues

The following section introduces the asymptotic way of solving control problems for many-server queues in the QED regime. It shows that, when scaled and centered around static fluid level (2.8), the queue process will asymptotically behave like a controlled diffusion with a drift control, which also partially explains the intuition behind the estimate (2.11). Given a cost, one obtains a diffusion control problem that can sometimes be solved explicitly and suggests a so-called asymptotically optimal solution to the original problem. In our case, however, a solution is not readily available, hence we present the MDP solution in Chapter 3. Nevertheless, very useful insights can be derived from the diffusion representation. Namely, the nature of the controls is to regulate the distribution of queues and idleness among the classes and stations; besides, an important dimension reduction, sometimes called state-space collapse, is observed in the pool-dependent networks.

2.3.1 Original model

Let $Y_i^n(t)$ and $Z_j^n(t)$ be, respectively, the number of class- i customers in the queue and the number of idle servers in pool j , at time t . Denote by A_i^n the arrival process of class- i customers, which is a Poisson process of rate λ_i^n . Let S_{ij}^n be a Poisson process with rate μ_{ij}^n and assume that the processes S_{ij}^n are mutually independent and independent of the arrival processes A_i^n . Similarly, let R_i^n be Poisson processes of rate θ_i^n , which are independent of each other and of the processes A_i^n and S_{ij}^n .

Let $\Psi_{ij}^n(t)$ be the number of class- i customers served in pool j at time t . Using the additivity of Poisson processes, the number of service completions in the interval

$(0, t]$ of class- i customers by pool j can be represented $S_{ij}^n(\int_0^t \Psi_{ij}^n(u)du)$. Likewise, since $\int_0^t Y_i^n(u)du$ indicates the time up to t that class- i customers wait in the queue, $R_i^n(\int_0^t Y_i^n(u)du)$ is equal, in the distribution, to the number of abandonments of class- i customers by time t . Thus we have,

$$X_i^n(t) = X_i^n(0) + A_i^n(t) - \sum_{j \in \mathcal{J}} S_{ij}^n \left(\int_0^t \Psi_{ij}^n(u)du \right) - R_i^n \left(\int_0^t Y_i^n(u)du \right), \quad i \in \mathcal{I} \quad (2.19)$$

as well as

$$X_i^n(t) = Y_i^n(t) + \sum_{j \in \mathcal{J}} \Psi_{ij}^n(t), \quad i \in \mathcal{I} \quad (2.20)$$

$$N_j^n = Z_j^n(t) + \sum_{i \in \mathcal{I}} \Psi_{ij}^n(t), \quad j \in \mathcal{J} \quad (2.21)$$

$$Y_i^n(t) \geq 0, \quad Z_j^n(t) \geq 0, \quad \Psi_{ij}^n(t) \geq 0, \quad i \in \mathcal{I}, \quad j \in \mathcal{J}, \quad t \geq 0. \quad (2.22)$$

The processes $\Psi^n = (\Psi_{ij}^n)_{(i,j) \in \mathcal{I} \times \mathcal{J}}$ are regarded as *scheduling control policy* and assumed to be right-continuous, taking values in $\mathbb{Z}_+$. We are interested in finding an optimal policy that minimizes the operational cost

$$V^{n,\pi}(x) = \mathbb{E}_x^\pi \left[\int_0^\infty e^{-\alpha t} L(Y^n(t)) dt \right] \quad (2.23)$$

for some function $L : \mathbb{R}^I \rightarrow \mathbb{R}$, over all possible policies $\pi = \{\Psi\}$.

Note that our problem is introduced (2.19) – (2.23) with preemptive control policies in mind. A policy is regarded as preemptive if the service to customers can be interrupted at any time and resumed at a later time, possibly in a different server pool. In contrast, under a non-preemptive policy, every customer must complete service with the server that is first assigned to the customer. Interestingly, these two versions are asymptotically equivalent in the QED regime. We refer to [14] for further explanations.

2.3.2 Diffusion model

We introduce the following scaling to capture the fluctuations around the static fluid:

$$\hat{A}_i^n(t) = \frac{A_i^n(t) - \lambda_i^n t}{\sqrt{n}}, \quad \hat{S}_{ij}^n(t) = \frac{S_{ij}^n(nt) - n\mu_{ij}^n t}{\sqrt{n}}, \quad \hat{R}_i^n(t) = \frac{R_i^n(nt) - n\theta_i^n t}{\sqrt{n}}. \quad (2.24)$$

Similarly, the processes representing states of the queueing system are centered about the static fluid model (2.8) and rescaled as below:

$$\hat{X}_i^n(t) = \frac{X_i^n(t) - nx_i^*}{\sqrt{n}}, \quad (2.25)$$

$$\hat{Y}_i^n(t) = \frac{Y_i^n(t)}{\sqrt{n}}, \quad \hat{Z}_j^n(t) = \frac{Z_j^n(t)}{\sqrt{n}}, \quad (2.26)$$

$$\hat{\Psi}_{ij}^n(t) = \frac{\Psi_{ij}^n(t) - \psi_{ij}^* n}{\sqrt{n}}. \quad (2.27)$$

We also assume that there are constants x_i , $i \in \mathcal{I}$, such that the initial conditions satisfy

$$\hat{X}_i^n(0) = \frac{X_i^n(0) - nx_i^*}{\sqrt{n}} \rightarrow x_i. \quad (2.28)$$

As a result, the relations (2.19) - (2.22) can be presented in the new form:

$$\hat{X}_i^n(t) = \hat{X}_i^n(0) + \hat{W}_i^n(t) - \sum_{j \in \mathcal{J}} \mu_{ij} \int_0^t \hat{\Psi}_{ij}^n(u) du - \theta_i^n \int_0^t \hat{Y}_i^n(u) du \quad (2.29)$$

$$\hat{Y}_i^n(t) + \sum_{j \in \mathcal{J}} \hat{\Psi}_{ij}^n(t) = \hat{X}_i^n(t), \quad (2.30)$$

$$\hat{Z}_j^n(t) + \sum_{i \in \mathcal{I}} \hat{\Psi}_{ij}^n(t) = 0, \quad (2.31)$$

$$\hat{Y}_i^n(t) \geq 0, \quad \hat{Z}_j^n(t) \geq 0, \quad i \in \mathcal{I}, j \in \mathcal{J}, t \geq 0 \quad (2.32)$$

where

$$\begin{aligned} \hat{W}_i^n(t) = & \hat{A}_i^n(t) - \sum_{j \in \mathcal{J}} \hat{S}_{ij}^n \left(\frac{1}{n} \int_0^t \Psi_{ij}^n(u) du \right) - \hat{R}_i^n \left(\frac{1}{n} \int_0^t Y_i^n(u) du \right) + \\ & \left(\frac{\lambda_i^n}{\sqrt{n}} - \sqrt{n} \sum_{j \in \mathcal{J}} \mu_{ij}^n \psi_{ij}^* \right) t. \end{aligned} \quad (2.33)$$

It can be shown, for each i , that regardless of the scheduling policy, $\hat{W}_i^n$ converges weakly to a Brownian motion $\tilde{W}_i(t)$ with a certain constant drift and variance. Therefore, by formally taking limits as $n \rightarrow \infty$ and denoting the weak limit of

$(\hat{X}^n, \hat{Y}^n, \hat{Z}^n, \hat{\Psi}^n, \hat{W}^n)$ by (X, Y, Z, Ψ, W) , we obtain diffusion equations as below.

$$X_i(t) = x_i + \tilde{W}_i(t) - \sum_{j \in \mathcal{J}} \mu_{ij} \int_0^t \Psi_{ij}(u) du - \theta_i \int_0^t Y_i(u) du, \quad i \in \mathcal{I} \quad (2.34)$$

$$\sum_{j \in \mathcal{J}} \Psi_{ij}(t) = X_i(t) - Y_i(t), \quad i \in \mathcal{I} \quad (2.35)$$

$$\sum_{i \in \mathcal{I}} \Psi_{ij}(t) = -Z_j(t), \quad j \in \mathcal{J} \quad (2.36)$$

$$Y_i(t) \geq 0, \quad Z_j(t) \geq 0, \quad i \in \mathcal{I}, \quad j \in \mathcal{J} \quad (2.37)$$

Similar to (2.23), for a certain policy $\pi = \{\Psi\}$, we introduce the cost function

$$V^\pi(x) = \mathbb{E}_x^\pi \int_0^\infty e^{-\alpha t} L(Y(t)) dt. \quad (2.38)$$

Atar, Mandelbaum and Shaikhet showed in [17] that the solution to the diffusion control problem $\sup_{\pi \in \Pi} V^\pi(x)$ dictates the optimal policy for the pre-limit control problem $\sup_{\pi \in \Pi} V^{n,\pi}(x)$.

2.3.3 Dimension reduction of pool-dependent models

For pool-dependent models, it was shown in [17] that the set of all possible policies can be restricted to work-conserving policies, i.e., the policies satisfying

$$Z_e^n \wedge Y_e^n \equiv 0. \quad (2.39)$$

In such a case, we can observe an interesting dimension reduction [18]. In particular, the controlled process (generally I -dimensional) becomes one-dimensional, with the total number of customers $\sum_{i \in \mathcal{I}} X_i$ being the only quantity of interest, which we elaborate below.

By summing equations (2.34) over all i 's, we obtain

$$X_e(t) = x_e + \tilde{W}_e(t) + \sum_{j \in \mathcal{J}} \mu_j \int_0^t Z_j(u) du - \sum_{i \in \mathcal{I}} \theta_i \int_0^t Y_i(u) du. \quad (2.40)$$

From (2.35) and (2.36), we get $X_e(t) = Y_e(t) - Z_e(t)$. Then, using (2.26), (2.32) and

(2.39), $Y_e(t)$ and $Z_e(t)$ can be represented as

$$Y_e(t) = X_e^+(t), \quad Z_e(t) = X_e^-(t). \quad (2.41)$$

After that, we can define a process $U = (w, v)$ taking values in

$$\mathcal{A} = \{(w, v) \in \mathbb{R}^I \times \mathbb{R}^J : w_i, v_j \geq 0, w_e = v_e = 1\}, \quad (2.42)$$

so that for all $t \geq 0$,

$$Y_i(t) = w_i(t)X_e^+(t), \quad Z_j(t) = v_j(t)X_e^-(t). \quad (2.43)$$

Intuitively, w represents the fraction of total queue length distributed in each class, while v indicates the fraction of total idleness assigned to each server pool. Rewrite (2.40):

$$X_e(t) = x_e + \tilde{W}_e(t) + \int_0^t [\mu \cdot v(u)] X_e^-(u) du - \int_0^t [\theta \cdot w(u)] X_e^+(u) du. \quad (2.44)$$

Together with the rewritten version of (2.38),

$$V^\pi(x_e) = \mathbb{E}_{x_e}^\pi \int_0^\infty e^{-\alpha t} L(w(t) \cdot X_e^+(t)) dt \quad (2.45)$$

we get a diffusion control problem $\sup_\pi V^\pi(x_e)$, where the action will depend on the sum X_e only.

Given the values of $X_e(t)$, as well as $w(t), v(t)$, the scheduling policy $\{\Psi(t)\}$ can be uniquely determined from (2.35) - (2.36). In what follows we explicitly derive the controlled diffusions for our networks of interest.

V model

In a two-class V model, $\mathcal{I} = \{1, 2\}$ and $\mathcal{J} = \{1\}$:

$$Y_1(t) = w_1(t)X_e^+(t), \quad Y_2(t) = w_2(t)X_e^+(t), \quad (2.46)$$

$$Z_1(t) = X_e^-(t), \quad (2.47)$$

$$\Psi_{11}(t) = X_1(t) - Y_1(t), \quad \Psi_{21}(t) = X_2(t) - Y_2(t), \quad \Psi_{11}(t) + \Psi_{21}(t) = -Z_1(t). \quad (2.48)$$

As a result, the equation (2.44) is simplified to

$$X_e(t) = x_e + W_e(t) + \mu \int_0^t X_e^-(u) du - \int_0^t (\theta_1 w_1(u) + \theta_2 w_2(u)) X_e^+(u) du. \quad (2.49)$$

N model

In a N model, $\mathcal{I} = \{1, 2\}$ and $\mathcal{J} = \{1, 2\}$. Assume that pool 1 serves customers of class 1, and pool 2 serves customers of both classes. In this case for any $t \geq 0$,

$$Y_1(t) = w_1(t) X_e^+(t), \quad Y_2(t) = w_2(t) X_e^+(t), \quad (2.50)$$

$$Z_1(t) = v_1(t) X_e^-(t), \quad Z_2(t) = v_2(t) X_e^-(t), \quad (2.51)$$

$$\Psi_{11}(t) + \Psi_{12}(t) = X_1(t) - Y_1(t), \quad \Psi_{22}(t) = X_2(t) - Y_2(t), \quad (2.52)$$

$$\Psi_{12}(t) + \Psi_{22}(t) = -Z_2(t), \quad \Psi_{11}(t) = -Z_1(t). \quad (2.53)$$

Consequently, the equation (2.44) is simplified to

$$X_e(t) = x_e + W_e(t) + \int_0^t (\mu_1 v_1(u) + \mu_2 v_2(u)) X_e^-(u) du - \int_0^t (\theta_1 w_1(u) + \theta_2 w_2(u)) X_e^+(u) du. \quad (2.54)$$

Further simplifications of the control space

It has been shown in [17], using the comparison's principle, that if the cost function does not include idleness X_e^- , taking $v_{j^*} \equiv 1$ (and $v_j \equiv 0$ for all $j \neq j^*$) for $j^* = \arg \min\{\mu_j, j \in \mathcal{J}\}$ is optimal for idleness allocation control v .

In Chapter 3, we will investigate the aforementioned control problem by applying Markov decision processes. It will be shown that, in the QED heavy traffic regime, the optimal policy indeed depends on the value of the sum X_e^n only. We will also gain some intuition regarding the form of the optimal policy.

Chapter 3

Markov Decision Process

3.1 Introduction

A Markov decision process (MDP) is a controlled stochastic process that satisfies the Markov property with costs associated to each state transition. A Markov decision problem consists of a MDP and a performance criterion. A solution to a Markov decision problem is called a policy, which determines the optimal control actions according to the performance criterion. MDP has been applied to model and solve decision-making problems under uncertainty in various fields, such as operations research, economics, combinatorial optimization, and the social sciences [27].

In this chapter, we first briefly review the theories of MDP, along with the algorithms often used to solve a Markov decision problem. Then, we present the optimal policies for control problems of V model, N model and X model by using MDP.

3.1.1 Preliminaries

A Markov decision process is characterized by four elements:

$$\{ S, A, q(j|i, a), c(i, a) \}. \quad (3.1)$$

Here state space S is the set of all states of the system; action space A is the set of all possible actions; $q(j|i, a)$ is the transition rate from state i to state j under action a ; $c(i, a)$ is the cost for taking action a at state i . In this chapter, we focus on finite state and action space and nonnegative cost functions.

A policy is a mapping from states to actions. If the policy is independent of time, it is called stationary. Let $\mathcal{F}$ denote the set of stationary policies.

Several performance criteria are commonly used to evaluate the policy, such as the finite-horizon expected cost, the infinite-horizon expected discount cost and the long-run expected average cost [28]. We are mainly interested in the infinite-horizon expected discount cost: for each policy $f \in \mathcal{F}$ and each initial state $x(0) = i$,

$$V_\alpha(i, f) = \mathbb{E}_i^f \left[\int_0^\infty e^{-\alpha t} c(x(t), f) dt \right], \quad (3.2)$$

where $x(t)$ is the state of the system at time t and $\alpha > 0$ is the discount factor.

Then the corresponding Markov decision problem is defined as below. Determine a policy $f^* \in \mathcal{F}$ such that

$$V_\alpha(i, f^*) = \min_{f \in \mathcal{F}} V_\alpha(i, f) =: V_\alpha^*(i) \quad \forall i \in S. \quad (3.3)$$

The basis for some practical algorithms that used to solve Markov decision problems is the optimality equation introduced in the following theorems (see [28] for more details).

Theorem 3.1. *The optimal discount cost $V_\alpha^*(i)$ satisfies the discount-cost optimality equation below,*

$$V_\alpha^*(i) = \min_{a \in A(i)} \left\{ \frac{c(i, a)}{\alpha + q_i(a)} + \frac{1}{\alpha + q_i(a)} \sum_{j \neq i} V_\alpha^*(j) q(j|i, a) \right\} \quad \forall i \in S, \quad (3.4)$$

where $q_i(a) = -q(i|i, a) \geq 0$ and $A(i)$ is the set of actions available when the state of the system is $i \in S$.

Theorem 3.2. *Suppose that the optimal discount cost $V_\alpha^*(i) < \infty$ for all $i \in S$. Then optimality equation (3.4) is equivalent to the following optimality equation:*

$$\alpha V_\alpha^*(i) = \min_{a \in A(i)} \{ c(i, a) + \sum_{j \in S} V_\alpha^*(j) q(j|i, a) \} \quad \forall i \in S. \quad (3.5)$$

3.1.2 Value iteration algorithm

The value iteration algorithm is a successive approximation algorithm for Markov decision problem devised by Bellman [29]. Define an operator T as follows: for any

nonnegative function u on S ,

$$Tu(i) = \min_{a \in A(i)} \left\{ \frac{c(i, a)}{\alpha + q_i(a)} + \frac{1}{\alpha + q_i(a)} \sum_{j \neq i} u(j)q(j|i, a) \right\} \quad \forall i \in S \quad (3.6)$$

Theorem 3.3. *Assume that continuity-compactness condition holds. Let $U_0^* = 0$ and $U_{n+1}^* = TU_n^*$ for $n \geq 0$. Then,*

1. $\{U_n^*\}$ is nondecreasing in $n \geq 0$, and $\lim_{n \rightarrow \infty} U_n^* = V_\alpha^*$.
2. Let $f_n \in \mathcal{F}(n \geq 0)$ be such that

$$\begin{aligned} U_{n+1}^*(i) &= \min_{a \in A(i)} \left\{ \frac{c(i, a)}{\alpha + q_i(a)} + \frac{1}{\alpha + q_i(a)} \sum_{j \neq i} U_n^*(j)q(j|i, a) \right\} \\ &= \frac{c(i, f_n(i))}{\alpha + q_i(f_n(i))} + \frac{1}{\alpha + q_i(f_n(i))} \sum_{j \neq i} U_n^*(j)q(j|i, f_n(i)) \end{aligned} \quad (3.7)$$

for all $i \in S$. Then the limit point of $\{f_n\}$ is a discount-cost optimal stationary policy.

The continuity-compactness condition (Assumption 4.12, page 64 in [28]) guarantees the existence of a discount-cost optimal policy.

Theorem 3.3 suggests the cost functions that computed by iterations based on the value operator T converge to the optimal cost function in the limit. In addition, the policies that associated with the successive cost functions also converge to the optimal policy in a finite number of iterations (please refer to [30] for detailed discussion).

The *value iteration algorithm* is described as follows:

1. Set $k = 0$ and take vector $V_k = 0$.
2. Loop:
 - (a) Obtain $V_{k+1}(i) = \min_{a \in A(i)} \left\{ \frac{c(i, a)}{\alpha + q_i(a)} + \frac{1}{\alpha + q_i(a)} \sum_{j \neq i} V_k(j)q(j|i, a) \right\}$ for all $i \in S$.
 - (b) If $\max_{i \in S} [V_{k+1}(i) - V_k(i)]$ is sufficiently small, stop; otherwise, increment k by 1 and repeat the loop.
3. Return policy $\{a \in A(i) : i \in S\}$ for all $i \in S$.

3.1.3 Policy iteration algorithm

Another well-known algorithm for solving MDPs is designed by Howard [31] and known as policy iteration. There are two phases involved in the policy iteration algorithm: policy evaluation and policy improvement. First, we introduce a theorem since it provides a method of evaluating policy.

Theorem 3.4. *For a finite Markov decision problem, $V_\alpha(f)$ is a unique bounded solution to the equation*

$$\alpha u(i) = c(i, f) + \sum_{j \in S} u(j)q(j|i, f) \quad \forall i \in S \quad (3.8)$$

for every $f \in \mathcal{F}$.

Next, we define an improvement policy. For every given $f \in \mathcal{F}$, $i \in S$, and $a \in A(i)$, let

$$D_f(i, a) = c(i, a) + \sum_{j \in S} V_\alpha(j, f)q(j|i, a) \quad (3.9)$$

and

$$E_f(i) = \{a \in A(i) : D_f(i, a) < \alpha V_\alpha(i, f)\}. \quad (3.10)$$

Then, an improvement policy $h \in \mathcal{F}$ is defined as follows:

$$h(i) \in E_f(i) \quad \text{if } E_f(i) \neq \emptyset \quad \text{and} \quad h(i) = f(i) \quad \text{if } E_f(i) = \emptyset \quad (3.11)$$

The following theorem explains the reason that policy h is an improvement of f .

Theorem 3.5. *For any given $f \in \mathcal{F}$, let $h \in \mathcal{F}$ be defined as in (3.11) and $h \neq f$. Then $V_\alpha(f) \succeq V_\alpha(h)$.*

We state the *policy iteration algorithm* as below:

1. Select an arbitrary $f \in \mathcal{F}$. Set $k = 0$ and take $f_k = f$.
2. Loop:
 - (a) (Policy evaluation) Obtain $V_\alpha(f_k)$ by solving (3.8), which is equal to $V_\alpha(f_k) = [\alpha I - Q(f_k)]^{-1}c(f_k)$.
 - (b) (Policy improvement) Obtain an improvement policy of f_k from (3.11) and set as f_{k+1} .

- (c) If $f_{k+1} = f_k$, stop; otherwise, increment k by 1 and repeat the loop.
3. Return policy f_k .

3.2 Optimal control of V model

In this section, we study the control problem of a Markovian two-class V model (see Figure 3.1) by interpreting it as a Markov decision problem. In this model, we assume that the arrival rate of class- i customers is λ_i ; the abandonment rate of class- i customers is θ_i ; the service rates of both classes are μ , because they are pool-dependent; the single server pool consists of N statistically identical and independent servers ($i \in \mathcal{I} = \{1, 2\}$).

Figure 3.1: V model

Let c_1 and c_2 denote the holding cost rates of class-1 and class-2 customers, respectively. The cost per unit time at time t is defined as $c_1 q_1(t) + c_2 q_2(t)$, where $q_i(t)$ represents the number of queued class- i customers at time t . There are a variety of ways to schedule the waiting customers. Our concern is to find the policy that minimizes the infinite-horizon expected discount cost $V_\alpha(x_1, x_2) = \mathbb{E}[\int_0^\infty e^{-\alpha t} (c_1 q_1(t) + c_2 q_2(t)) dt]$.

The solution to this control problem is obvious when $c_1 > c_2$ and $\theta_1 < \theta_2$. It would be optimal to hold class-2 customers in the queue because they are cheaper and more likely to abandon. However, when $c_1 > c_2$ and $\theta_1 > \theta_2$, one must make a trade-off between the low cost rate and high abandonment rate. In other words, if

one maintains waiting population in class 1, the high abandonment rate is helpful to reduce the cost but the high cost rate is unfavorable; similarly, if one keeps customers of class 2 waiting, one will enjoy the low cost rate but also have to accept the low abandonment rate.

The explicit solution of this problem by using diffusion control model has not been provided. We interpret the problem as a Markov decision problem and apply the policy iteration algorithm to solve the optimal control policy. The algorithm was implemented in MATLAB, and the code is given in Appendix [A.1](#).

3.2.1 Markov decision problem

First of all, we specify the basic elements [\(3.1\)](#) of the MDP associated with the above V model.

State space

Let $x_i(t)$ denote the total number of class- i customers in the system at time t ($i \in \{1, 2\}$). The state of the system at time t can be described by a two-dimensional vector $(x_1(t), x_2(t))$. To simulate a queueing system with infinite waiting capacity, the upper bounds of x_1 and x_2 are set as $N + O(N^{2/3})$, which was shown in [\(2.14\)](#) that would be reached with very small probability in the QED regime. So we denote by S the state space and set

$$S = \{(x_1, x_2) | 0 \leq x_i \leq N + O(N^{2/3}), \quad i \in \mathcal{I}\}. \quad (3.12)$$

Action space

Considering preemptive, work-conserving and stationary policies for this control problem, we define (s_1, s_2) as a control action of state (x_1, x_2) . s_i indicates the number of class- i customers in service ($i \in \mathcal{I}$). Then, $x_i - s_i$ is the queue length of class- i customers. Let $A(x_1, x_2)$ denote the set of available actions at state $(x_1, x_2) \in S$.

$$A(x_1, x_2) = \{(s_1, s_2) | s_1 + s_2 = (x_1 + x_2) \wedge N, 0 \leq s_i \leq x_i, i \in \mathcal{I}\} \quad (3.13)$$

In Chapter 2, we show by [\(2.49\)](#) that this control problem can be simplified to one-dimensional, with $x_1 + x_2$, the total number of customers, being the only quantity

of interest. In other words, the control policy can be uniquely determined when given $x_1 + x_2$ and (w_1, w_2) , the fraction of total queued customers. By definition,

$$w_1 = 1 - w_2 = \frac{x_1 - s_1}{x_1 + x_2 - N} \mathbb{1}_{\{x_1 + x_2 > N\}}, \quad (3.14)$$

where $s_1 \in [0 \vee (N - x_2), x_1 \wedge N]$.

Note that in the QED heavy traffic regime, x_1 fluctuates within $[nx_1^* - n^{\frac{1}{2}+\gamma}, nx_1^* + n^{\frac{1}{2}+\gamma}]$, which indicates $x_1 \wedge N = x_1$. Likewise, $0 \vee (N - x_2) = N - x_2$. Therefore, $0 \leq w_1 \leq 1$ and its extreme values can always be achieved.

Transition rates

The transition rate from state (x_1, x_2) to state (x'_1, x'_2) under action $a = (s_1, s_2)$ are represented as follows:

$$q((x'_1, x'_2)|(x_1, x_2), a = (s_1, s_2)) = \begin{cases} \lambda_1 & \text{if } x'_1 = x_1 + 1, x'_2 = x_2, \\ \lambda_2 & \text{if } x'_1 = x_1, x'_2 = x_2 + 1, \\ \mu_1 s_1 + (x_1 - s_1)\theta_1 & \text{if } x'_1 = x_1 - 1, x'_2 = x_2, \\ \mu_2 s_2 + (x_2 - s_2)\theta_2 & \text{if } x'_1 = x_1, x'_2 = x_2 - 1, \\ -\lambda_1 - \lambda_2 - \mu_1 s_1 - (x_1 - s_1)\theta_1 - \mu_2 s_2 - (x_2 - s_2)\theta_2 & \text{if } x'_1 = x_1, x'_2 = x_2, \\ 0 & \text{otherwise.} \end{cases} \quad (3.15)$$

Cost function

Considering linear holding cost, we denote c_i the cost of keeping a class- i customer waiting for a unit time ($i \in \mathcal{I}$). Then, for policy $f = \{(s_1(t), s_2(t))\}$, cost per unit time at time t is

$$c(x_1(t), x_2(t), f) = c_1(x_1(t) - s_1(t)) + c_2(x_2(t) - s_2(t)). \quad (3.16)$$

We also need to specify the performance criterion of the Markov decision problem.

Performance criterion

For each policy $f \in \mathcal{F}$ and each initial state $(x_1, x_2) \in S$, the infinite-horizon expected discount cost is

$$V_\alpha(x_1, x_2, f) = \mathbb{E}\left[\int_0^\infty e^{-\alpha t} c(x_1(t), x_2(t), f) dt\right]. \quad (3.17)$$

Our goal is to determine a policy $f^* \in \mathcal{F}$ such that

$$V_\alpha(x_1, x_2, f^*) = \min_{f \in \mathcal{F}} V_\alpha(x_1, x_2, f) := V_\alpha^*(x_1, x_2). \quad (3.18)$$

3.2.2 Numerical analysis

When the parameters of the above model are taken appropriate values, the optimal policy can be obtained by using MATLAB programs. We set parameters as follows.

Parameters Setting

- Discount factor $\alpha = 0.9$.
- Consider the QED regime, we set $\mu = 6$, $N = 50$ and $\lambda_1 = \lambda_2 = 150$.
- $\theta_1 = 5$, $\theta_2 = 1$. We need to ensure $\max(\theta_1, \theta_2) \leq \mu$, because otherwise it may be optimal to let customers abandon rather than to provide them service.
- $c_1 = 11$, $c_2 = 5$.
- Set the state space $S = \{(x_1, x_2) | 0 \leq x_1, x_2 \leq 80\}$, in which $80 > N + 2N^{2/3} \approx 77$.

To better visualize the optimal policy generated by the programs, we mark the states showing different control actions with different shapes and colors.

As shown in Figure 3.2 (a), the x-axis and y-axis represent x_1 and x_2 , respectively. In other words, each point represents a state $(x_1, x_2) \in S$. For states $\{(x_1, x_2) | x_1 + x_2 \leq N\}$, $\{(s_1, s_2) | s_1 = x_1, s_2 = x_2\}$ is the only available action, so it is unnecessary to concern oneself with their optimal actions. We mark these states with blue "+". For states with s_1 taken its maximum $x_1 \wedge N$, w_1 is taken its minimum, so the optimal action is to keep class-2 customers waiting. We mark such states with pink "o".

Figure 3.2: Optimal policy for V model: $c_1 = 11$

Likewise, for states with s_1 taken its minimum $0 \vee (N - x_2)$, the optimal action is to hold class-1 customers in the queue. We mark these states with red “*”.

As suggested by the claim (2.11) in Chapter 2, a rectangle window around the static fluid model should be set as our observation region:

$$\{(x_1, x_2) | x_1 \in [nx_1^* - n^{\frac{1}{2}+\gamma_1}, nx_1^* + n^{\frac{1}{2}+\gamma_1}], x_2 \in [nx_2^* - n^{\frac{1}{2}+\gamma_2}, nx_2^* + n^{\frac{1}{2}+\gamma_2}]\}. \quad (3.19)$$

With our parameters, $n = 50$, $x_1^* = x_2^* = 0.5$.

Besides, we found the optimal discount costs $V_\alpha^*(x_1, x_2)$ of the states having the same $x_1 + x_2$ are very close to each other. So we plot their relationship in Figure 3.2 (b), in which the x-axis and y-axis are $x_1 + x_2$ and $V_\alpha^*(x_1, x_2)$, respectively.

The optimal policy depends on $x_1 + x_2$

As can be seen, Figure 3.2-3.5 are plotted under different cost rate c_1 . Figure 3.3 (a) shows that, when c_1 is small, the optimal policy is to keep waiting population in class 1. This is intuitively clear because it is more economical to keep the less patient

(a) Optimal policy

(b) Relation between $V_{\alpha}^*(x_1, x_2)$ and $x_1 + x_2$ **Figure 3.3:** Optimal policy for V model: $c_1 = 7$

(a) Optimal policy

(b) Relation between $V_{\alpha}^*(x_1, x_2)$ and $x_1 + x_2$ **Figure 3.4:** Optimal policy for V model: $c_1 = 14$

Figure 3.5: Optimal policy for V model: $c_1 = 16$

customers waiting if they are very cheap to hold.

However, if the cost rate c_1 becomes higher, the optimal policy indicates that one should keep waiting population in class 2 at some states (pink region) and to class 2 at the others (red region). Within the observation region, there is a boundary separating the pink and red regions, which is similar to a straight line $x_1 + x_2 = k$ ($k \geq N$). As suggested by Figure 3.2 and 3.4, k becomes bigger when c_1 increases.

If c_1 is sufficiently big (see Figure 3.5), the optimal policy is to hold class-2 customers in the queue. Obviously, it is better to serve class-1 customers as soon as possible if their holding cost is much higher than class 2, no matter how often they abandon the system.

To sum up, one can conclude that the optimal policy depends on the total number of customers $x_1 + x_2$ only. There is a threshold value of $x_1 + x_2$ that determines the change of optimal action. Moreover, the threshold value is related to the cost rates c_1 and c_2 .

The optimal discount cost depends on $x_1 + x_2$

As shown in Figure 3.2 (b) - 3.5 (b), the optimal discount costs $V_\alpha^*(x_1, x_2)$ with the same $x_1 + x_2$ converge to one point in large scale, which indicates that $V_\alpha^*(x_1, x_2)$ is probably a function of $x_1 + x_2$.

In addition, the colors representing different optimal action separate well, which also demonstrates that the optimal policy depends on $x_1 + x_2$. For example, in Figure 3.2 (b), for states $\{(x_1, x_2) | 50 < x_1 + x_2 < 60\}$, the optimal action is to keep class-2 customers waiting (pink region); for states $\{(x_1, x_2) | x_1 + x_2 \geq 60\}$, it is optimal to hold class-1 customers in the queue (red region).

Relation between $\frac{dV_\alpha^*(x_1, x_2)}{d(x_1 + x_2)}$ and $\frac{c_1 - c_2}{\theta_1 - \theta_2}$

According to the conjecture proposed by Shaikh et al. [18], the optimal policy is decided by the relation between $\frac{c_1 - c_2}{\theta_1 - \theta_2}$ and the derivative of $V_\alpha^*(x_1, x_2)$ with respect to $x_1 + x_2$. To check the conjecture, we take the minimum among the costs with the same $x_1 + x_2$ as a representative of $V_\alpha^*(x_1, x_2)$ and calculate the derivative as below:

$$\frac{dV_\alpha^*(x_1, x_2)}{d(x_1 + x_2)} := V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1). \quad (3.20)$$

In Figure 3.6, x-axis represents $x_1 + x_2$. We take $x_1 + x_2 > N$ because our main concern is the distribution of waiting customers. The blue lines describe the derivatives under different c_1 s, while the red lines indicate the corresponding $\frac{c_1 - c_2}{\theta_1 - \theta_2}$.

By comparing Figure 3.6 and Figure 3.2 - 3.5, one can find an accordance between them. For example, in Figure 3.6 (b), the derivative exceeds $\frac{c_1 - c_2}{\theta_1 - \theta_2} = 1.50$ when $x_1 + x_2$ reaches 60; while in Figure 3.2, the threshold value of $x_1 + x_2$ is also 60. Therefore, the optimal policy can be described as follows: when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) \leq \frac{c_1 - c_2}{\theta_1 - \theta_2}$, it is optimal to keep waiting population in class 2; when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) > \frac{c_1 - c_2}{\theta_1 - \theta_2}$, it is optimal to keep waiting population in class 1.

In addition, the slight decrease of the derivatives at the end of the blue line in Figure 3.6 (d) suggests that the optimal expected cost $V_\alpha^*(x_1, x_2)$ is not necessary a convex function of $x_1 + x_2$. As a result, there might be more than one threshold in the optimal policy.

Figure 3.6: V model: relation between $\frac{dV_\alpha^*(x_1, x_2)}{d(x_1 + x_2)}$ and $\frac{c_1 - c_2}{\theta_1 - \theta_2}$

3.2.3 Proof

In this section, we prove that the optimal policy for the above V model is “bang-bang”; that is, the control variable w_1 (or equivalently, s_1) can only take its extreme values. Decisions regarding whether to take the maximum or the minimum depend on the relation between $\frac{c_1 - c_2}{\theta_1 - \theta_2}$ and $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1)$.

Theorem 3.6. *For a pool-dependent V model, the optimal expected cost $V_\alpha^*(x_1, x_2)$ is a nondecreasing function of the total number of customers $x_1 + x_2$.*

- When $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) \leq \frac{c_1 - c_2}{\theta_1 - \theta_2}$, it is optimal to keep waiting population in the class i^* , where $i^* = \arg \min\{\theta_i, i \in \mathcal{I}\}$;
- When $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) > \frac{c_1 - c_2}{\theta_1 - \theta_2}$, it is optimal to keep waiting population in the class i^* , where $i^* = \arg \max\{\theta_i, i \in \mathcal{I}\}$.

Proof: First, we show that $V_\alpha^*(x_1, x_2)$ is a non-decreasing function of $x_1 + x_2$. From (3.14), the control action $(s_1(t), s_2(t))$ is uniquely determined by $x_1(t) + x_2(t)$ when given $(w_1(t), w_2(t))$; that is to say, the dynamic of controlling the system depends on the sum $x_1(t) + x_2(t)$ rather than $x_1(t)$ and $x_2(t)$. Besides, by (3.14) and (3.16),

$$c(x_1(t), x_2(t), f) = [c_1 w_1(t) + c_2(1 - w_1(t))](x_1(t) + x_2(t) - N) \mathbb{1}_{\{x_1(t) + x_2(t) > N\}}. \quad (3.21)$$

So the cost associated to each state transition depends only the sum $x_1(t) + x_2(t)$ as well. Thus, the optimal discount cost $V_\alpha^*(x_1, x_2) = \min_{f \in \mathcal{F}} \mathbb{E}[\int_0^\infty e^{-\alpha t} c(x_1(t), x_2(t), f) dt]$ is a non-decreasing function of $x_1 + x_2$, the sum of initial state; that is,

$$V_\alpha^*(x_1 - 1, x_2) = V_\alpha^*(x_1, x_2 - 1) := V_\alpha^*(x_1 + x_2 - 1). \quad (3.22)$$

According to Theorem 3.2, for each state $(x_1, x_2) \in S$,

$$\begin{aligned} \alpha V_\alpha^*(x_1, x_2) = \min_{(s_1, s_2) \in A(x_1, x_2)} \{ & \sum_{(x'_1, x'_2) \in S} V_\alpha^*(x'_1, x'_2) q[(x'_1, x'_2) | (x_1, x_2), a = (s_1, s_2)] \\ & + c_1(x_1 - s_1) + c_2(x_2 - s_2) \}. \end{aligned} \quad (3.23)$$

By substituting the transition rates (3.15),

$$\begin{aligned} \alpha V_\alpha^*(x_1, x_2) = & \min_{(s_1, s_2) \in A(x_1, x_2)} \{ c_1(x_1 - s_1) + c_2(x_2 - s_2) + \lambda_1 V_\alpha^*(x_1 + 1, x_2) \\ & + \lambda_2 V_\alpha^*(x_1, x_2 + 1) + [\mu s_1 + \theta_1(x_1 - s_1)] V_\alpha^*(x_1 - 1, x_2) \\ & + [\mu s_2 + \theta_2(x_2 - s_2)] V_\alpha^*(x_1, x_2 - 1) - [\lambda_1 + \lambda_2 + \mu s_1 \\ & + (x_1 - s_1)\theta_1 + \mu s_2 + (x_2 - s_2)\theta_2] V_\alpha^*(x_1, x_2) \} \\ = & \min_{(s_1, s_2) \in A(x_1, x_2)} \{ C + s_1[(\mu - \theta_1)(V_\alpha^*(x_1 + x_2 - 1) - V_\alpha^*(x_1, x_2)) - c_1] \\ & + s_2[(\mu - \theta_2)(V_\alpha^*(x_1 + x_2 - 1) - V_\alpha^*(x_1, x_2)) - c_2] \}, \end{aligned} \quad (3.24)$$

where C is the part independent of s_1 and s_2 .

By (3.13), $s_2 = (x_1 + x_2) \wedge N - s_1$. Then, for states $\{(x_1, x_2) | x_1 + x_2 \leq N\}$, $s_2 = x_2$; for states $\{(x_1, x_2) | x_1 + x_2 > N\}$, $s_2 = N - s_1$.

When $x_1 + x_2 \leq N$, $(s_1, s_2) = (x_1, x_2)$ is the only available action. It is unnecessary to consider its optimal action.

When $x_1 + x_2 > N$, by substituting $s_2 = N - s_1$ in (3.24),

$$\alpha V_\alpha^*(x_1, x_2) = \min_{\substack{(s_1, s_2) \in \\ A(x_1, x_2)}} \{C' + s_1[(\theta_1 - \theta_2)(V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1)) - (c_1 - c_2)]\}. \quad (3.25)$$

where $C' = C + N[(\mu - \theta_2)(V_\alpha^*(x_1, x_2) - 1) - c_2]$.

Notice that

$$V_\alpha^*(x_1 + x_2) \geq V_\alpha^*(x_1 + x_2 - 1). \quad (3.26)$$

Therefore,

- if $\theta_1 < \theta_2$,
 1. when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) \leq \frac{c_1 - c_2}{\theta_1 - \theta_2}$, one should take the minimum of s_1 , i.e. $s_1 = (N - x_2) \vee 0$, which means to take the maximum of w_1 . Thus, it is optimal to keep waiting population in class 1;
 2. when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) > \frac{c_1 - c_2}{\theta_1 - \theta_2}$, one should take the maximum of s_1 , i.e. $s_1 = x_1 \wedge N$, which means to take the minimum of w_1 . Thus, it is optimal to keep waiting population in class 2.
- if $\theta_1 > \theta_2$,
 1. when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) \leq \frac{c_1 - c_2}{\theta_1 - \theta_2}$, one should take the maximum of s_1 ; that is, it is optimal to keep waiting population in class 2;
 2. when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) > \frac{c_1 - c_2}{\theta_1 - \theta_2}$, one should take the minimum of s_1 ; that is, it is optimal to keep waiting population in class 1.

3.3 Optimal control of N model

A pool-dependent N model is a direct extension of the two-class V model studied in the previous section. As shown in Figure 3.7, pool 1, which consists of N_1 servers, serves class-1 customers with rate μ_1 ; pool 2, which contains N_2 servers, serve customers of both class 1 and 2 with the same rate μ_2 .

For this N model, we are also interested in determining the optimal policy to minimize $V_\alpha(x_1(t), x_2(t), f) = E[\int_0^\infty e^{-\alpha t} c(x_1(t), x_2(t), f) dt]$, the expected discount

Figure 3.7: N model

cost of keeping customers waiting. The Markov decision problem associated with this queueing model is similar to that of the above V model. It can be solved by a modified policy iteration algorithm, for which the code is given in Appendix [A.2](#).

3.3.1 Markov decision problem

State space

As suggested by (2.16) and (2.17) in Chapter 2, $N_1 + N_2$ and N_2 are the asymptotic upper bounds of the number of class-1 customers and the number of class-2 customers, respectively. Hence, to simulate a queueing system with infinite waiting capacity, we define the state space

$$S = \{(x_1, x_2) | 0 \leq x_1 \leq N_1 + N_2, 0 \leq x_2 \leq N_2\}. \quad (3.27)$$

Action space

Considering preemptive, work-conserving and stationary policies, we designate the following set $A(x_1, x_2)$ as the action space for state $(x_1, x_2) \in S$:

$$A(x_1, x_2) = \{(s_{11}, s_{12}, s_{22}) | s_{11} + s_{12} + s_{22} = (x_1 + x_2) \wedge (N_1 + N_2), \\ 0 \leq s_{ij} \leq x_i, i \in \mathcal{I}, j \in \mathcal{J}\}. \quad (3.28)$$

Here s_{ij} indicates the number of class- i customers served by pool j .

Notice that the simplified one-dimensional diffusion equation (2.54) contains two control variables $\{(w, v) \in \mathbb{R}^I \times \mathbb{R}^J : w_i, v_j \geq 0, w_e = v_e = 1\}$. w and v represent the ratios of total queued customers and total idle servers, respectively. According to work-conservation, $s_{11} = N_1$ when $x_1 + x_2 > N_1 + N_2$, so

$$w_1 = 1 - w_2 = \frac{x_1 - N_1 - s_{12}}{x_1 + x_2 - (N_1 + N_2)} \mathbb{1}_{\{x_1 + x_2 > N_1 + N_2\}}, \quad (3.29)$$

where $s_{12} \in [N_2 - x_2, (x_1 - N_1) \wedge N_2]$. And

$$v_1 = 1 - v_2 = \frac{N_1 - s_{11}}{N_1 + N_2 - (x_1 + x_2)} \mathbb{1}_{\{x_1 + x_2 < N_1 + N_2\}}, \quad (3.30)$$

where $s_{11} \in [(x_1 + x_2 - N_2) \vee 0, N_1 \wedge x_1]$.

Note that in the QED heavy traffic regime, x_1 fluctuates within $[nx_1^* - n^{\frac{1}{2}+\gamma}, nx_1^* + n^{\frac{1}{2}+\gamma}]$, which means $(x_1 - N_1) \wedge N_2 = x_1 - N_1$ and $N_1 \wedge x_1 = N_1$. Likewise, $(x_1 + x_2 - N_2) \vee 0 = x_1 + x_2 - N_2$. Therefore, $0 \leq w_1 \leq 1$, $0 \leq v_1 \leq 1$, and their extreme values can always be achieved.

Transition rates

The transition rate from state (x_1, x_2) to state (x'_1, x'_2) under action $a = (s_{11}, s_{12}, s_{22})$ is

$$q((x'_1, x'_2)|(x_1, x_2), a = (s_{11}, s_{12}, s_{22})) = \begin{cases} \lambda_1 & \text{if } x'_1 = x_1 + 1, x'_2 = x_2, \\ \lambda_2 & \text{if } x'_1 = x_1, x'_2 = x_2 + 1, \\ \mu_1 s_{11} + \mu_2 s_{12} + (x_1 - s_{11} - s_{12})\theta_1 & \text{if } x'_1 = x_1 - 1, x'_2 = x_2, \\ \mu_2 s_{22} + (x_2 - s_{22})\theta_2 & \text{if } x'_1 = x_1, x'_2 = x_2 - 1, \\ \rho & \text{if } x'_1 = x_1, x'_2 = x_2, \\ 0 & \text{otherwise.} \end{cases} \quad (3.31)$$

where $\rho = -\lambda_1 - \lambda_2 - \mu_1 s_{11} - \mu_2(s_{12} + s_{22}) - (x_1 - s_{11} - s_{12})\theta_1 - (x_2 - s_{22})\theta_2$.

Cost function

For policy $f = \{(s_{11}(t), s_{12}(t), s_{22}(t))\}$, cost per unit time at time t is

$$c(x_1(t), x_2(t), f) = c_1(x_1(t) - s_{11}(t) - s_{12}(t)) + c_2(x_2(t) - s_{22}(t)). \quad (3.32)$$

3.3.2 Numerical analysis

In our MATLAB programs, parameters are set as below.

Parameters Setting

- Discount factor $\alpha = 0.9$.
- Consider the QED regime, $\mu_1 = 6$, $\mu_2 = 8$, $N_1 = 50$, $N_2 = 50$, $\lambda_1 = 500$ and $\lambda_2 = 200$.
- $\theta_1 = 5$, $\theta_2 = 1$.
- $c_1 = 11$, $c_2 = 5$.
- The state space is set as $S = \{(x_1, x_2) | 0 \leq x_1 \leq 100, 0 \leq x_2 \leq 50\}$.

Same as above, states are highlighted with several shapes and colors in the figures we obtained. For states $\{(x_1, x_2) | x_1 + x_2 \leq N_1 + N_2\}$, the state will be marked with green "x" if its action is to take the minimum of s_{11} , which indicates to keep idleness in pool 1; the state will be marked with black "x" if the action is taking the maximum of s_{11} , which indicates to keep idleness in pool 2; the state will be marked with blue "+" if the action is neither of the above two. For states $\{(x_1, x_2) | x_1 + x_2 > N_1 + N_2\}$, the state will be marked with pink "o" if s_{12} is taken the maximum, that is, one should keep class-2 customers waiting for being served by pool 2; the state will be marked with red "*" if s_{12} is taken the minimum, that is, one should keep class-1 customers waiting for being served by pool 2; the state will be marked with blue "+" if s_{12} is not taken its extreme values.

The rectangle windows in the following figures indicate the observation regions, which, as suggested by claim (2.11), should be around the static fluid model (2.8).

Figure 3.8: Optimal policy for N model: $c_1 = 6$

Figure 3.9: Optimal policy for N model: $c_1 = 8$

Idleness is kept in the slower server pool

As can be seen in Figure 3.8-3.11, the optimal action of states $\{(x_1, x_2) | x_1 + x_2 \leq N_1 + N_2\}$ is to keep idle servers in pool 1. It is intuitively clear that when there are free servers in both pools, customers would choose the faster servers. With our parameters, servers in pool 2 are faster and will be kept busy. So idle servers should be kept in pool 1.

Figure 3.10: Optimal policy for N model: $c_1 = 9$

Figure 3.11: Optimal policy for N model: $c_1 = 12$

The optimal policy depends on $x_1 + x_2$

For states $\{(x_1, x_2) | x_1 + x_2 > N_1 + N_2\}$, there is a boundary separating states with different optimal actions. Within the observation region, the boundary is similar to a straight line $x_1 + x_2 = k$. When c_1 increases, k becomes bigger. For example, when $c_1 = 9$ (see Figure 3.10), the optimal policy for states $\{(x_1, x_2) | 100 < x_1 + x_2 < 106\}$ is approximately keeping class-2 customers waiting for available pool-2 servers; while for states $\{(x_1, x_2) | x_1 + x_2 \geq 106\}$, it is optimal to hold the waiting population in class 1. Above all, one can conjecture that the optimal policy is decided by $x_1 + x_2$, the

Figure 3.12: N model: relation between $\frac{dV_{\alpha}^*(x_1, x_2)}{d(x_1 + x_2)}$ and $\frac{c_1 - c_2}{\theta_1 - \theta_2}$

total number of customers in the system. In addition, the threshold value of $x_1 + x_2$ depends on the cost rates c_1 and c_2 .

Relation between $\frac{c_1 - c_2}{\theta_1 - \theta_2}$ and $\frac{dV_{\alpha}^*(x_1, x_2)}{d(x_1 + x_2)}$

Similarly to the above V model, the optimal expected cost $V_{\alpha}^*(x_1, x_2)$ of N model is shown to be a function of $x_1 + x_2$. We calculate the derivatives $\frac{dV_{\alpha}^*(x_1, x_2)}{d(x_1 + x_2)} := V_{\alpha}^*(x_1 + x_2) - V_{\alpha}^*(x_1 + x_2 - 1)$ and compare them with $\frac{c_1 - c_2}{\theta_1 - \theta_2}$. Since our main concern is the distribution of queued customers, we only plot the derivatives when $x_1 + x_2 > N_1 + N_2$ in Figure 3.12.

The blue line shows the derivatives $\frac{dV_{\alpha}^*(x_1, x_2)}{d(x_1 + x_2)}$ with respect to $x_1 + x_2 \in [101, 143]$, while the red line indicates the value of $\frac{c_1 - c_2}{\theta_1 - \theta_2}$. Figure 3.12 (a), (b) and (d) coincide with Figure 3.8, 3.9 and 3.11. They demonstrate that when $V_{\alpha}^*(x_1 + x_2) - V_{\alpha}^*(x_1 +$

$x_2 - 1) \leq \frac{c_1 - c_2}{\theta_1 - \theta_2}$, it is optimal to keep waiting population in class 2; when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) > \frac{c_1 - c_2}{\theta_1 - \theta_2}$, the optimal action is to keep waiting customers in class 1. Figure 3.12(c) does not completely agree with Figure 3.10 because when $x_1 + x_2 > 130$, the derivatives become less than $\frac{c_1 - c_2}{\theta_1 - \theta_2}$, but Figure 3.10 does not show the change of optimal policy. We consider this an error due to the restriction of MDP programs. Figure 3.12 also suggests that $V_\alpha^*(x_1, x_2)$ is not necessary a convex function of $x_1 + x_2$.

3.3.3 Proof

In this section, we provide a proof to show the “bang-bang” policy presented above is optimal for the control of N model. The policy includes two parts: first, it is optimal to keep idle servers in pool 1, that is, to take the maximum of control variable v_1 (or equivalently, the minimum of s_{11}); second, one should keep waiting population either in class 1 or class 2, that is, w_1 (or equivalently, s_{12}) should only take its extreme values. Whether to take the maximum or minimum of w_1 (or s_{12}) is decided by the relation between $\frac{c_1 - c_2}{\theta_1 - \theta_2}$ and $\frac{dV_\alpha^*(x_1, x_2)}{d(x_1 + x_2)}$.

Theorem 3.7. *For a pool-dependent N model, the optimal expected cost $V_\alpha^*(x_1, x_2)$ is a non-decreasing function of the total number of customers $x_1 + x_2$.*

- When $x_1 + x_2 \leq N_1 + N_2$, it is optimal to keep idleness in pool j^* , where $j^* = \arg \min\{\mu_j, j \in \mathcal{J}\}$.
- When $x_1 + x_2 > N_1 + N_2$,
 1. if $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) \leq \frac{c_1 - c_2}{\theta_1 - \theta_2}$, it is optimal to keep waiting population in the class i^* , where $i^* = \arg \min\{\theta_i, i \in \mathcal{I}\}$;
 2. if $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) > \frac{c_1 - c_2}{\theta_1 - \theta_2}$, it is optimal to keep waiting population in the class i^* , where $i^* = \arg \max\{\theta_i, i \in \mathcal{I}\}$.

Proof: First, we show that $V_\alpha^*(x_1, x_2)$ is a non-decreasing function of $x_1 + x_2$. From (3.29) and (3.30), the control action $(s_{11}(t), s_{12}(t), s_{22}(t))$ can be uniquely determined by $x_1(t) + x_2(t)$ when given $w(t)$ and $v(t)$; that is to say, the dynamic of controlling the system depends on the sum $x_1(t) + x_2(t)$ rather than $x_1(t)$ and $x_2(t)$. Besides, by

(3.29) and (3.32),

$$c(x_1(t), x_2(t), f) = [c_1 w_1(t) + c_2(1 - w_1(t))](x_1(t) + x_2(t) - N_1 - N_2) \mathbb{1}_{\{x_1(t) + x_2(t) > N_1 + N_2\}}. \quad (3.33)$$

So the cost associated to each state transition depends only on the sum $x_1(t) + x_2(t)$ as well. Consequently, the optimal discount cost $V_\alpha^*(x_1, x_2) = \min_{f \in \mathcal{F}} \mathbb{E}[\int_0^\infty e^{-\alpha t} c(x_1(t), x_2(t), f) dt]$ is a non-decreasing function of $x_1 + x_2$, the sum of initial state; that is,

$$V_\alpha^*(x_1 - 1, x_2) = V_\alpha^*(x_1, x_2 - 1) := V_\alpha^*(x_1 + x_2 - 1). \quad (3.34)$$

From Theorem 3.2, for each state $(x_1, x_2) \in S$,

$$\begin{aligned} \alpha V_\alpha^*(x_1, x_2) = & \min_{\substack{(s_{11}, s_{12}, s_{22}) \in \\ A(x_1, x_2)}} \{c_1(x_1 - s_{11} - s_{12}) + c_2(x_2 - s_{22}) \\ & + \sum_{(x'_1, x'_2) \in S} V_\alpha^*(x'_1, x'_2) q[(x'_1, x'_2) | (x_1, x_2), a = (s_{11}, s_{12}, s_{22})]\} \end{aligned} \quad (3.35)$$

By transition rates (3.31),

$$\begin{aligned} \alpha V_\alpha^*(x_1, x_2) = & \min_{(s_{11}, s_{12}, s_{22}) \in A(x_1, x_2)} \{c_1 x_1 + c_2 x_2 - c_1 s_{11} - c_1 s_{12} - c_2 s_{22} + \lambda_1 V_\alpha^*(x_1 + 1, x_2) \\ & + \lambda_2 V_\alpha^*(x_1, x_2 + 1) + [\mu_1 s_{11} + \mu_2 s_{12} + \theta_1(x_1 - s_{11} - s_{12})] V_\alpha^*(x_1 - 1, x_2) \\ & + [\mu_2 s_{22} + \theta_2(x_2 - s_{22})] V_\alpha^*(x_1, x_2 - 1) - [\lambda_1 + \lambda_2 + \mu_1 s_{11} + \mu_2(s_{12} + s_{22}) \\ & + \theta_1(x_1 - s_{11} - s_{12}) + \theta_2(x_2 - s_{22})] V_\alpha^*(x_1, x_2)\} \\ = & \min_{(s_{11}, s_{12}, s_{22}) \in A(x_1, x_2)} \{C + s_{11}[(\mu_1 - \theta_1)(V_\alpha^*(x_1 - 1, x_2) - V_\alpha^*(x_1, x_2)) - c_1] \\ & + s_{12}[(\mu_2 - \theta_1)(V_\alpha^*(x_1 - 1, x_2) - V_\alpha^*(x_1, x_2)) - c_1] \\ & + s_{22}[(\mu_2 - \theta_2)(V_\alpha^*(x_1, x_2 - 1) - V_\alpha^*(x_1, x_2)) - c_2]\} \end{aligned} \quad (3.36)$$

where C is the part independent of s_{11} , s_{12} , and s_{22} .

- When $x_1 + x_2 < N_1 + N_2$, because of work-conservation and $x_2 < N_2$,

$$s_{22} = x_2 \quad \text{and} \quad s_{12} = x_1 - s_{11}. \quad (3.37)$$

By substituting (3.37) in (3.36),

$$\alpha V_{\alpha}^*(x_1, x_2) = \min_{\substack{(s_{11}, s_{12}, s_{22}) \in \\ A(x_1, x_2)}} \{C' + s_{11}[(\mu_1 - \mu_2)(V_{\alpha}^*(x_1 + x_2 - 1) - V_{\alpha}^*(x_1 + x_2))]\} \quad (3.38)$$

in which, $C' = C + [x_1(\mu_2 - \theta_1) + x_2(\mu_2 - \theta_2)](V_{\alpha}^*(x_1 + x_2 - 1) - V_{\alpha}^*(x_1 + x_2)) - x_1c_1 - x_2c_2$ is also independent of the control variables.

Because $V_{\alpha}^*(x_1 + x_2 - 1) - V_{\alpha}^*(x_1 + x_2) \leq 0$,

1. if $\mu_1 < \mu_2$, one should take the minimum of s_{11} , i.e. the maximum of v_1 . Thus, it is optimal to keep idleness in pool 1.
2. if $\mu_1 > \mu_2$, one should take the maximum of s_{11} , i.e. the minimum of v_1 . Thus, it is optimal to keep idleness in pool 2.

- When $x_1 + x_2 \geq N_1 + N_2$, in consequence of work-conservation,

$$s_{11} = x_1 \quad \text{and} \quad s_{12} + s_{22} = N_2. \quad (3.39)$$

By substituting (3.39) in (3.36),

$$\alpha V_{\alpha}^*(x_1, x_2) = \min_{\substack{(s_{11}, s_{12}, s_{22}) \in \\ A(x_1, x_2)}} \{C' + s_{12}[(\theta_1 - \theta_2)(V_{\alpha}^*(x_1 + x_2) - V_{\alpha}^*(x_1 + x_2 - 1)) - (c_1 - c_2)]\} \quad (3.40)$$

in which $C' = C + [x_1(\mu_1 - \theta_1) + N_2(\mu_2 - \theta_2)](V_{\alpha}^*(x_1 + x_2 - 1) - V_{\alpha}^*(x_1 + x_2)) - c_1x_1 - c_2N_2$ is independent of the control variables.

Thus, to achieve the minimum of $\alpha V_{\alpha}^*(x_1, x_2)$,

1. if $\theta_1 < \theta_2$,
 - (a) when $V_{\alpha}^*(x_1 + x_2) - V_{\alpha}^*(x_1 + x_2 - 1) \leq \frac{c_1 - c_2}{\theta_1 - \theta_2}$, one should take the minimum of s_{12} , i.e. $s_{12} = N_2 - x_2$, which means to take the maximum of w_1 . Thus, it is optimal to keep waiting population in class 1;
 - (b) when $V_{\alpha}^*(x_1 + x_2) - V_{\alpha}^*(x_1 + x_2 - 1) > \frac{c_1 - c_2}{\theta_1 - \theta_2}$, one should take the maximum of s_{12} , i.e. $s_{12} = (x_1 - N_1) \wedge N_2$, which means to take the minimum of w_1 . Thus, it is optimal to keep waiting population in class 2;

2. if $\theta_1 > \theta_2$,
 - (a) when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) \leq \frac{c_1 - c_2}{\theta_1 - \theta_2}$, one should take the maximum of s_{12} ; that is, it is optimal to keep waiting population in class 2;
 - (b) When $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) > \frac{c_1 - c_2}{\theta_1 - \theta_2}$, one should take the minimum of s_{12} , that is, it is optimal to keep waiting population in class 1.

3.4 Optimal control of X model

X model with pool-dependent service rates is another extension of the two-class V model we discussed. Unlike the N model, customers of both classes are able to get service from either of the two pools. As shown in Figure 3.13, pool 1 serves both class-1 and class-2 customers with rate μ_1 , while pool 2 servers both classes with rate μ_2 .

Figure 3.13: X model

For this X model, we are still interested in finding the optimal policy that can minimize $E[\int_0^\infty e^{-\alpha t} c(x_1(t), x_2(t), f) dt]$, the expected discount cost of holding customers in queue. This control problem is once again interpreted as a Markov decision problem in this section. We observe the optimal policy obtained by using the policy iteration algorithm, for which the code is presented in Appendix A.3.

3.4.1 Markov decision problem

State space

As suggested by (2.18) in Chapter 2, $N_1^n + N_2^n + O((N_1^n + N_2^n)^{\frac{2}{3}})$ is an asymptotically upper bound of the total number of customers. Hence, we define the state space as

$$S = \{(x_1, x_2) | 0 \leq x_i \leq N_1^n + N_2^n + O((N_1^n + N_2^n)^{\frac{2}{3}}), \quad i \in \mathcal{I}\}. \quad (3.41)$$

Action space

Considering preemptive, work-conserving and stationary policies, we define the action space for state $(x_1, x_2) \in S$ by

$$A(x_1, x_2) = \{(s_{11}, s_{21}, s_{12}, s_{22}) | s_{11} + s_{21} + s_{12} + s_{22} = (x_1 + x_2) \wedge (N_1 + N_2), \\ 0 \leq s_{ij} \leq x_i, \quad i \in \mathcal{I}, \quad j \in \mathcal{J}\}, \quad (3.42)$$

where s_{ij} denotes the number of class- i customers served by pool j .

Let $w = (w_1, w_2)$ denote the ratios of queued class-1 and class-2 customers. Let $v = (v_1, v_2)$ be the ratios of idle servers in pool 1 and pool 2. Then we have,

$$w_1 = 1 - w_2 = \frac{x_1 - s_{11} - s_{12}}{x_1 + x_2 - N_1 - N_2} \mathbb{1}_{\{x_1 + x_2 > N_1 + N_2\}}, \quad (3.43)$$

where $s_{11} + s_{12} \in [(N_1 + N_2 - x_2) \vee 0, x_1 \wedge (N_1 + N_2)]$;

$$v_1 = 1 - v_2 = \frac{N_1 - s_{11} - s_{21}}{N_1 + N_2 - x_1 - x_2} \mathbb{1}_{\{x_1 + x_2 < N_1 + N_2\}}, \quad (3.44)$$

where $s_{11} + s_{21} \in [(x_1 + x_2 - N_2) \vee 0, (x_1 + x_2) \wedge N_1]$.

According to (2.12), $x_1 + x_2$ fluctuates within $[n(\nu_1 + \nu_2) - n^{\frac{1}{2}+\gamma}, n(\nu_1 + \nu_2) + n^{\frac{1}{2}+\gamma}]$ in the QED regime. It indicates that $x_i < N_1 + N_2$ for $i \in \mathcal{I}$, as well as $N_j < x_1 + x_2$ for $j \in \mathcal{J}$. Therefore, $0 \leq w_1 \leq 1$, $0 \leq v_1 \leq 1$, and their extreme values can always be achieved.

Transition rates

The transition rate from state (x_1, x_2) to state (x'_1, x'_2) under action $a = (s_{11}, s_{21}, s_{12}, s_{22})$ is

$$q((x'_1, x'_2)|(x_1, x_2), a = (s_{11}, s_{21}, s_{12}, s_{22})) = \begin{cases} \lambda_1 & \text{if } x'_1 = x_1 + 1, x'_2 = x_2, \\ \lambda_2 & \text{if } x'_1 = x_1, x'_2 = x_2 + 1, \\ \mu_1 s_{11} + \mu_2 s_{12} + (x_1 - s_{11} - s_{12})\theta_1 & \text{if } x'_1 = x_1 - 1, x'_2 = x_2, \\ \mu_1 s_{21} + \mu_2 s_{22} + (x_2 - s_{21} - s_{22})\theta_2 & \text{if } x'_1 = x_1, x'_2 = x_2 - 1, \\ \rho & \text{if } x'_1 = x_1, x'_2 = x_2, \\ 0 & \text{otherwise.} \end{cases} \quad (3.45)$$

where $\rho = -\lambda_1 - \lambda_2 - \mu_1(s_{11} + s_{21}) - \mu_2(s_{12} + s_{22}) - (x_1 - s_{11} - s_{12})\theta_1 - (x_2 - s_{21} - s_{22})\theta_2$.

Cost function

For policy $f = \{(s_{11}(t), s_{21}(t), s_{12}(t), s_{22}(t))\}$, cost per unit time at time t is

$$c(x_1(t), x_2(t), f) = c_1(x_1(t) - s_{11}(t) - s_{12}(t)) + c_2(x_2(t) - s_{21}(t) - s_{22}(t)). \quad (3.46)$$

3.4.2 Numerical analysis

In our MATLAB programs, parameters are chosen as follows.

Parameters Setting

- Discount factor $\alpha = 0.9$.
- Consider the QED regime, $\mu_1 = 6$, $\mu_2 = 8$, $N_1 = N_2 = 20$, and $\lambda_1 = \lambda_2 = 140$.
- $\theta_1 = 5$, $\theta_2 = 1$.
- $c_1 = 10$, $c_2 = 5$.
- The state space is set as $S = \{(x_1, x_2) | 0 \leq x_1 \leq 70, 0 \leq x_2 \leq 70\}$. Because $N_1 + N_2 + 2(N_1 + N_2)^{\frac{2}{3}} < 70$.

In the following figures, states are highlighted with the same shape and color as in the previous section. For states marked with green "x", the optimal action is to take the minimum of $s_{11} + s_{21}$ (or equivalently, the maximum of v_1); that is, it is optimal to keep idle servers in pool 1. For states marked with pink "o", the optimal action

is to take the minimum of $s_{11} + s_{12}$ (or equivalently, the maximum of w_1); that is, one should keep waiting population in class 2. For states marked with red "*", the optimal action is to take the maximum of $s_{11} + s_{12}$ (or equivalently, the minimum of w_1); that is, one should keep waiting population in class 1. For states marked with blue "+", none of above actions is optimal.

The rectangle windows in the following figures indicate the observation regions. Although, for the pool-dependent X model, there is no unique optimal solution to the static fluid model (2.8), by claim (2.12) we can enlarge the observation region to

$$\{(x_1, x_2) | x_i \in [0, n(v_1 + v_2) + n^{\frac{1}{2}+\gamma}], \quad \forall i \in \mathcal{I}\}. \quad (3.47)$$

where $n = 20$ and $\nu_1 = \nu_2 = 1$, with our parameters.

Idleness is kept in the slower server pool

The green triangles in Figure 3.14 suggest it is optimal to keep idle servers in pool 1, of which the service rate is smaller. This is intuitively true because it is more efficient for customers to choose the faster servers when there are available servers in both pools.

The optimal policy depends on $x_1 + x_2$

For states $\{(x_1, x_2) | x_1 + x_2 > N_1 + N_2\}$, there are two types of optimal actions within the observation region. One is to keep class-2 customers waiting (pink region); the other is to hold class-1 customers in the queue (red region). The pink and red regions are separated approximately by a straight line $x_1 + x_2 = k$. With our parameters, the bigger c_1 is, the bigger k is. For instance, Figure 3.14(b) shows that if $c_1 = 12$, the optimal policy is to keep waiting population in class 2 when $40 < x_1 + x_2 \leq 54$ and in class 1 when $x_1 + x_2 > 54$. Meanwhile, in Figure 3.14(c), the threshold value of $x_1 + x_2$ is . To sum up, the optimal policy depends on $x_1 + x_2$.

Relation between $\frac{c_1 - c_2}{\theta_1 - \theta_2}$ and $\frac{dV_\alpha^*(x_1, x_2)}{d(x_1 + x_2)}$

The relation between derivatives $\frac{dV_\alpha^*(x_1, x_2)}{d(x_1 + x_2)} := V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1)$ and $\frac{c_1 - c_2}{\theta_1 - \theta_2}$ is plotted in Figure 3.15.

(a) $c_1 = 7$ (b) $c_1 = 12$ (c) $c_1 = 13$ (d) $c_1 = 15$ **Figure 3.14:** Optimal policy for X model

The x-axis represents $x_1 + x_2$, for which we take $x_1 + x_2 > N_1 + N_2$ because that is the case when there exist queued customers. The blue lines describe the derivatives of $V_\alpha^*(x_1, x_2)$ with respect to $x_1 + x_2$; the red lines show the values $\frac{c_1 - c_2}{\theta_1 - \theta_2}$. One can find Figure 3.15 in accordance with Figure 3.14. For example, in Figure 3.15(b),

Figure 3.15: X model: relation between $\frac{dV_\alpha^*(x_1, x_2)}{d(x_1 + x_2)}$ and $\frac{c_1 - c_2}{\theta_1 - \theta_2}$

the derivatives are less than $\frac{c_1 - c_2}{\theta_1 - \theta_2}$ when $x_1 + x_2 \leq 55$ and bigger than $\frac{c_1 - c_2}{\theta_1 - \theta_2}$ when $55 < x_1 + x_2 < 70$; while in Figure 3.14, 55 is also the threshold value separating the pink and red regions. This coincidence suggests that the optimal policy is to keep class-2 customers waiting when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) \leq \frac{c_1 - c_2}{\theta_1 - \theta_2}$; it is optimal to hold class-1 customers in the queue when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) > \frac{c_1 - c_2}{\theta_1 - \theta_2}$. Besides, Figure 3.15 (b), (c) and (d) imply that the optimal expected cost $V_\alpha^*(x_1, x_2)$ of X model, same as that of N model, is not necessary a convex function of $x_1 + x_2$.

3.4.3 Proof

In the following, we prove the optimal policy for the control of X model is "bang-bang", which means the control variables w and v should take only their extreme values. In the language of queueing systems, it is optimal to keep idle servers in pool

1 and waiting population in either class 1 or class 2. Whether to give priority to class 1 or class 2 is decided by the relationship between $\frac{c_1 - c_2}{\theta_1 - \theta_2}$ and $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1)$.

Theorem 3.8. *For a pool-dependent X model, the optimal expected cost $V_\alpha^*(x_1, x_2)$ is a non-decreasing function of the total number of customers $x_1 + x_2$.*

- When $x_1 + x_2 \leq N_1 + N_2$, it is optimal to keep idleness in pool j^* , where $j^* = \arg \min\{\mu_j, j \in \mathcal{J}\}$;
- When $x_1 + x_2 > N_1 + N_2$,
 1. if $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) \leq \frac{c_1 - c_2}{\theta_1 - \theta_2}$, it is optimal to keep waiting population in the class i^* , where $i^* = \arg \min\{\theta_i, i \in \mathcal{I}\}$;
 2. if $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) > \frac{c_1 - c_2}{\theta_1 - \theta_2}$, it is optimal to keep waiting population in the class i^* , where $i^* = \arg \max\{\theta_i, i \in \mathcal{I}\}$;

Proof: First, we show that $V_\alpha^*(x_1, x_2)$ is a non-decreasing function of $x_1 + x_2$. From (3.43) and (3.44), the control action of our interest can be uniquely determined by $x_1(t) + x_2(t)$ when given $w(t)$ and $v(t)$; that is, the dynamic of controlling the system depends on the sum $x_1(t) + x_2(t)$ rather than $x_1(t)$ and $x_2(t)$. Besides, by (3.43) and (3.46),

$$c(x_1(t), x_2(t), f) = [c_1 w_1(t) + c_2(1 - w_1(t))](x_1(t) + x_2(t) - N_1 - N_2) \mathbb{1}_{\{x_1(t) + x_2(t) > N_1 + N_2\}}. \quad (3.48)$$

So the cost associated to each transition depends only on the sum $x_1(t) + x_2(t)$ as well. Hence, the optimal discount cost $V_\alpha^*(x_1, x_2) = \min_{f \in \mathcal{F}} \mathbb{E}[\int_0^\infty e^{-\alpha t} c(x_1(t), x_2(t), f) dt]$ is a non-decreasing function of $x_1 + x_2$, the sum of initial state. That is to say,

$$V_\alpha^*(x_1 - 1, x_2) = V_\alpha^*(x_1, x_2 - 1) := V_\alpha^*(x_1 + x_2 - 1). \quad (3.49)$$

According to Theorem 3.2, for each state $(x_1, x_2) \in S$,

$$\begin{aligned} \alpha V_\alpha^*(x_1, x_2) = & \min_{\substack{(s_{11}, s_{21}, s_{12}, s_{22}) \in \\ A(x_1, x_2)}} \{c_1(x_1 - s_{11} - s_{12}) + c_2(x_2 - s_{21} - s_{22}) \\ & + \sum_{(x'_1, x'_2) \in S} V_\alpha^*(x'_1, x'_2) q[(x'_1, x'_2) | (x_1, x_2), a = (s_{11}, s_{21}, s_{12}, s_{22})]\} \end{aligned} \quad (3.50)$$

From transition rates (3.45),

$$\begin{aligned}
\alpha V_\alpha^*(x_1, x_2) &= \min_{(s_{11}, s_{21}, s_{12}, s_{22}) \in A(x_1, x_2)} \{c_1 x_1 + c_2 x_2 - c_1 s_{11} - c_1 s_{12} - c_2 s_{21} - c_2 s_{22} \\
&\quad + \lambda_1 V_\alpha^*(x_1 + 1, x_2) + \lambda_2 V_\alpha^*(x_1, x_2 + 1) + V_\alpha^*(x_1 - 1, x_2) [\mu_1 s_{11} + \mu_2 s_{12} \\
&\quad + \theta_1(x_1 - s_{11} - s_{12})] + V_\alpha^*(x_1, x_2 - 1) [\mu_1 s_{21} + \mu_2 s_{22} + \theta_2(x_2 - s_{21} - s_{22})] \\
&\quad - V_\alpha^*(x_1, x_2) [\lambda_1 + \lambda_2 + \mu_1(s_{11} + s_{21}) + \mu_2(s_{12} + s_{22}) + \theta_1(x_1 - s_{11} - s_{12}) \\
&\quad + \theta_2(x_2 - s_{21} - s_{22})]\} \\
&= \min_{(s_{11}, s_{21}, s_{12}, s_{22}) \in A(x_1, x_2)} \{s_{11}[(\mu_1 - \theta_1)(V_\alpha^*(x_1 + x_2 - 1) - V_\alpha^*(x_1 + x_2)) - c_1] \\
&\quad + s_{21}[(\mu_1 - \theta_2)(V_\alpha^*(x_1 + x_2 - 1) - V_\alpha^*(x_1 + x_2)) - c_2] \\
&\quad + s_{12}[(\mu_2 - \theta_1)(V_\alpha^*(x_1 + x_2 - 1) - V_\alpha^*(x_1 + x_2)) - c_1] \\
&\quad + s_{22}[(\mu_2 - \theta_2)(V_\alpha^*(x_1 + x_2 - 1) - V_\alpha^*(x_1 + x_2)) - c_2] + C\}
\end{aligned} \tag{3.51}$$

where C is the part independent of s_{11} , s_{21} , s_{12} , or s_{22} .

- When $x_1 + x_2 \leq N_1 + N_2$, because of work-conservation,

$$s_{11} + s_{12} = x_1 \quad \text{and} \quad s_{21} + s_{22} = x_2. \tag{3.52}$$

By substituting (3.52) in (3.51),

$$\alpha V_\alpha^*(x_1, x_2) = \min_{\substack{(s_{11}, s_{21}, s_{12}, s_{22}) \in \\ A(x_1, x_2)}} \{C' + (s_{11} + s_{21})(\mu_1 - \mu_2)(V_\alpha^*(x_1 + x_2 - 1) - V_\alpha^*(x_1 + x_2))\} \tag{3.53}$$

in which, $C' = C + [x_1(\mu_2 - \theta_1) + x_2(\mu_2 - \theta_2)](V_\alpha^*(x_1 + x_2 - 1) - V_\alpha^*(x_1 + x_2)) - c_1 x_2 - c_2 x_2$ is independent of the control variables.

Notice that $V_\alpha^*(x_1 + x_2 - 1) - V_\alpha^*(x_1 + x_2) \leq 0$,

1. if $\mu_1 < \mu_2$, one should take the minimum of $s_{11} + s_{21}$, i.e. $(x_1 + x_2 - N_2) \vee 0$, which indicates to take the maximum of v_1 . Thus, it is optimal to keep idle servers in pool 1;
2. if $\mu_1 > \mu_2$, one should take the maximum of $s_{11} + s_{21}$, i.e. $(x_1 + x_2) \wedge N_1$, which indicates to take the minimum of v_1 . Thus, it is optimal to keep idle servers in pool 2.

- When $x_1 + x_2 > N_1 + N_2$, due to work-conservation,

$$s_{11} + s_{21} = N_1 \quad \text{and} \quad s_{12} + s_{22} = N_2. \quad (3.54)$$

Then,

$$\begin{aligned} \alpha V_\alpha^*(x_1, x_2) = \min_{\substack{(s_{11}, s_{21}, s_{12}, s_{22}) \in \\ A(x_1, x_2)}} \{ & (s_{11} + s_{12})[(\theta_1 - \theta_2)(V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1)) \\ & - (c_1 - c_2)] + C' \} \end{aligned} \quad (3.55)$$

in which $C' = C + [N_1(\mu_1 - \theta_2) + N_2(\mu_2 - \theta_2)](V_\alpha^*(x_1 + x_2 - 1) - V_\alpha^*(x_1 + x_2)) - c_2(N_1 + N_2)$ is independent of the control variables.

Notice that $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) \geq 0$,

1. if $\theta_1 < \theta_2$,

- (a) when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) \leq \frac{c_1 - c_2}{\theta_1 - \theta_2}$, one should take the minimum of $s_{11} + s_{12}$, i.e. $(N_1 + N_2 - x_2) \vee 0$, which indicates to take the maximum of w_1 . Thus, it is optimal to keep waiting population in class 1;
- (b) when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) > \frac{c_1 - c_2}{\theta_1 - \theta_2}$, one should take the maximum of $s_{11} + s_{12}$, i.e. $x_1 \wedge (N_1 + N_2)$, which indicates to take the minimum of w_1 . Thus, it is optimal to keep waiting population in class 2.

2. if $\theta_1 > \theta_2$,

- (a) when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) \leq \frac{c_1 - c_2}{\theta_1 - \theta_2}$, one should take the maximum of $s_{11} + s_{12}$; that is, it is optimal to keep waiting population in class 2;
- (b) when $V_\alpha^*(x_1 + x_2) - V_\alpha^*(x_1 + x_2 - 1) > \frac{c_1 - c_2}{\theta_1 - \theta_2}$, one should take the minimum of $s_{11} + s_{12}$; that is, it is optimal to keep waiting population in class 1.

Chapter 4

Discrete-Event Simulation

4.1 Introduction

Simulation is an effective approach to analyzing complex systems. In queueing theory, simulation is often used to estimate the performance measures of queueing systems. For instance, Garnett and Mandelbaum [8] used simulation to evaluate different control policies and demonstrate their impact on the performance of system.

Queueing system $M/M/N$ can be easily simulated by applying the memoryless property of exponential random variables. However, simulation of the general system $G/G/N$ remains a challenging problem. Inspired by the algorithm of simulating $G/G/1$ in [22], we provide a simple algorithm for $G/G/N$ system. Furthermore, the simulation approach can be applied to complex queueing networks with general distributed interarrival time, service time and patience time. In this chapter, we first introduce the algorithm for $G/G/1$ given in [22], then elaborate our algorithm for $G/G/N$, finally present the algorithms for three extended examples, which are $G/G/N$ with abandonment, V model and N model.

Before going to the details, we introduce two essential components of discrete-event simulation: events and variables. In a queueing system, events usually include customers' arrival, service completion and, in some cases, abandonment. There are three types of variables often used to keep track of the system: time variable t , which is the amount of time elapsed since the start; state variables, which describe the state of the system at time t ; and counter variables, which count the number of certain events by time t . All the variables must be updated when an event takes place. We suggest to enlist the variables for observation.

Note that all below simulation algorithms are introduced with First-Come-First-Served (FCFS) discipline in mind, in which customers are served in the order of their arrivals. When there is more than one class of customers, we assume FCFS discipline within each customer class.

4.2 Simulation of G/G/1 model

In a G/G/1 queueing model (see Figure 4.1), we assume the distributions of interarrival time and service time are G_A and G_S , respectively. Customer's departure event indicates the service completion in this system.

Figure 4.1: G/G/1 model

The following variables are used to do the simulation:

- t : time variable;
- T : $T(i)$ is the occurring time of the i th event;
- X : $X(i)$ is the number of customers in system when the i th event happens;
- t_A : the next arrival time (after time t);
- t_D : the next departure time (after time t);
- N_A : the number of arrivals by time t ;
- N_D : the number of departures by time t ;

Let T_D denote the ending time of observation. Y_A and Y_S represent, respectively,

the random variables with G_A and G_S distributions. The algorithm for G/G/1 simulation is given as follows.

Initialization:

- Set $t = 0$, $T(0) = 0$, $X(0) = 0$, $N_A = N_D = 0$.
- Generate the time of first arrival $t_A = 0 + Y_A$.
- Set $t_D = T_D$.

t_D is set as T_D to ensure the first event in this empty system is an arrival.

There are two types of events. If $t_A \leq t_D$, the upcoming event is an arrival; otherwise, the upcoming event is a departure.

Loop:

- Case 1: $t_A \leq t_D$
 1. $t = t_A$ (move along with time t_A).
 2. $T(i) = t$ (record the occurring time of the i th event).
 3. $X(i) = X(i - 1) + 1$.
 4. $N_A = N_A + 1$.
 5. Generate a Y_A and reset $t_A = t + Y_A$, time of next arrival.
 6. (a) If $X(i) = 1$, generate a Y_S and reset $t_D = t + Y_S$.
Note: $X(i) = 1$ means the new customer gets service upon the arrival. Then, his/her service completion time needs to be generated as the next departure time in the system.
 - (b) If $X(i) > 1$, t_D remains the same.
Note: $X(i) > 1$ means the new customer joins the queue upon the arrival, so next departure time in the system will not be changed.
- Case 2: $t_A > t_D$
 1. $t = t_D$.
 2. $T(i) = t$.

3. $X(i) = X(i - 1) - 1$.

4. $N_D = N_D + 1$.

5. (a) If $X(i) = 0$, reset $t_D = T_D$.

Note: $X(i) = 0$ means the system is empty. t_D is set as T_D to ensure the next event is an arrival.

(b) If $X(i) > 0$, generate a Y_S and reset $t_D = t + Y_S$, time of next departure.

A loop is terminated when $\min(t_A, t_D) > T_D$.

Changes of the system's states can be observed by plotting $(T(i), X(i))$ for all i 's. Estimates of performance measures, for example average number of customers and average waiting time, can be calculated with T , X , N_A and N_D .

4.3 Simulation of G/G/N model

In a G/G/N queueing model (see Figure 4.2), there can be more than one server in the system. The challenge of simulating G/G/N systems is to follow up the status of the N servers. To solve the problem, a $N \times 1$ vector variable S is created, in which $S(k)$ represents the status of the k th server. Let $S(k)$ be the next service completion time if the k th server is busy; let $S(k) = T_D$ if the server is idle. Clearly, the next departure time in the system is the earliest service completion time among all the busy servers, which is indeed the minimum element of vector S .

Figure 4.2: G/G/N model

Assume that the service times of the N servers are independent and identically distributed according to G_S distribution. The distribution of interarrival time is G_A . Let Y_A and Y_S denote the random variables with G_A and G_S distributions, respectively. The algorithm for G/G/N simulation is shown as below. The code written in MATLAB is given in Appendix [B.1](#)

Initialization:

- Set $t = 0$, $T(0) = 0$, $X(0) = 0$, $S = (T_D)_{N \times 1}$, $N_A = N_D = 0$.
- Generate the time of first arrival $t_A = 0 + Y_A$.
- Set $t_D = \min(S) = T_D$.

Loop:

- Case 1: $t_A \leq t_D$
 1. $t = t_A$.
 2. $T(i) = t$.
 3. $X(i) = X(i - 1) + 1$.
 4. $N_A = N_A + 1$.
 5. Update t_A : generate a Y_A and reset $t_A = t + Y_A$.
 6. Update t_D :
 - (a) If $X(i) \leq N$, generate a Y_S , replace one of the T_D s in vector S with $t + Y_S$ and reset $t_D = \min(S)$.
Note: $X(i) \leq N$ means the new customer gets service upon the arrival. Hence one of the idle servers, who corresponds to a T_D in S , needs to be assigned to the new customer.
 - (b) If $X(i) > N$, S and t_D remain the same.
Note: $X(i) > N$ means the new customer joins the queue upon the arrival.
- Case 2: $t_A > t_D$
 1. $t = t_D$.
 2. $T(i) = t$.

3. $X(i) = X(i - 1) - 1$.
4. $N_D = N_D + 1$.
5. Update t_D :
 - (a) If $X(i) \geq N$, generate Y_S , replace the element t_D in S with $t + Y_S$ and reset $t_D = \min(S)$.
 Note: $X(i) \geq N$ means there were waiting customers before this departure. Then the server, who contributed to the departure, should serve a waiting customer now.
 - (b) If $X(i) < N$, change the element t_D in S to T_D and reset $t_D = \min(S)$.
 Note: $X(i) < N$ means there was no queue. Then the server, who contributed to the departure, should be idle now.

A loop is terminated when $\min(t_A, t_D) > T_D$.

4.4 Three extended examples

The core of the discrete-event simulation is to create variables to track the status of every customer and every server in the system. With these variables, the simulation can be implemented by clarifying the types of events and translating the scheduling and routing policies into the update of variables. This approach of simulation can be applied to queueing models with complex structures and various policies.

4.4.1 Simulation of G/G/N with abandonment

In a G/G/N queueing system with abandonment (see Figure 4.3), customers' departure events include service completion and abandonment. Consequently, the next departure event is the earlier one between the next service completion and the next abandonment.

To simulate the system, one need to track the patience time of every customer. Hence, a vector variable R is created, in which $R(j)$ represents the patience time of the j th arriving customer. Let $R(j) = T_D$ if the j th arriving customer is in service or has completed the service.

Denote by G_R the distribution of customers' patience time. Let Y_A , Y_S and Y_R be the random variables with G_A , G_S and G_R distributions, respectively. The simulation

Figure 4.3: G/G/N with abandonment

algorithm for G/G/N with abandonment is presented as follows. The code written in MATLAB is provided in [Appendix B.2](#)

Initialization:

- Set $t = 0$, $T(0) = 0$, $X(0) = 0$, $S = (T_D)_{N \times 1}$, $N_A = N_D = 0$.
- Generate the time of first arrival $t_A = 0 + Y_A$.
- Set $t_D = T_D$.

Loop:

- Case 1: $t_A \leq t_D$ (Arrival)
 1. $t = t_A$.
 2. $T(i) = t$.
 3. $X(i) = X(i - 1) + 1$.
 4. $N_A = N_A + 1$.
 5. Update t_A : generate a Y_A and reset $t_A = t + Y_A$.
 6. Update t_D :
 - ✓ if $X(i) \leq N$

(a) set $R(N_A) = T_D$.

Note: $X(i) \leq N$ means there are idle servers, so the newcomer gets service directly.

(b) generate a Y_S , replace one of the T_D s in vector S with $t + Y_S$.

(c) reset $t_D = \min(S) \wedge \min(R)$.

✓ if $X(i) > N$

(a) generate a Y_R and set $R(N_A) = t + Y_R$.

(b) S remains the same. Reset $t_D = \min(S) \wedge \min(R)$.

7. Check whether $t_D \in S$ or $t_D \in R$.

• Case 2: $t_A > t_D$ and $t_D \in S$ (Service completion)

1. $t = t_D$.

2. $T(i) = t$.

3. $X(i) = X(i - 1) - 1$.

4. $N_D = N_D + 1$.

5. Update t_D :

✓ if $X(i) \geq N$

(a) find the first non- T_D element in R and change it to T_D .

(b) generate a Y_S and replace the element t_D in S with $t + Y_S$.

(c) reset $t_D = \min(S) \wedge \min(R)$.

✓ if $X(i) < N$

(a) replace the element t_D in S with T_D .

(b) R remains the same. Reset $t_D = \min(S) \wedge \min(R)$.

6. Check whether $t_D \in S$ or $t_D \in R$.

• Case 3: $t_A > t_D$ and $t_D \in R$ (Abandonment)

1. $t = t_D$.

2. $T(i) = t$.

3. $X(i) = X(i - 1) - 1$.

4. $N_D = N_D + 1$.

5. Update t_D :
 - (a) find the element t_D in R and change it to T_D .
 - (b) S remains the same. Reset $t_D = \min(S) \wedge \min(R)$.
6. Check whether $t_D \in S$ or $t_D \in R$.

A loop is terminated when $\min(t_A, t_D) > T_D$.

4.4.2 Simulation of V model

To simulate a V model (see Figure 4.4), we first need to specify the scheduling and routing policy. Here we consider giving priority to class-1 customers, which means keeping class-2 customers waiting until there is no class-1 customer waiting.

Denote by G_{A_1} and G_{A_2} the distributions of interarrival time of class-1 and class-2 customers. The service time of class 1 and class 2 are distributed according to G_{S_1} and G_{S_2} distributions. Let Y_{A_1} , Y_{A_2} , Y_{S_1} and Y_{S_2} represent the random variables with G_{A_1} , G_{A_2} , G_{S_1} and G_{S_2} distributions, respectively.

Figure 4.4: V model

The following variables are introduced for the simulation:

- t_{A_k} : The next arrival time of class- k customers after time t ($k = 1, 2$);
- Q_k : The queue length of class- k customers at time t ($k = 1, 2$);

Then, time of next arrival event in the system is $t_A = \min(t_{A_1}, t_{A_2})$.

The simulation algorithm for V model is shown as below. The code written in MATLAB is given in Appendix [B.3](#)

Initialization:

- Set $t = 0$, $T(0) = 0$, $X(0) = 0$, $Q_k = 0$ ($k = 1, 2$), $S = (T_D)_{N \times 1}$, $N_A = N_D = 0$.
- Generate $t_{A_k} = 0 + Y_{A_k}$ ($k = 1, 2$), and set $t_A = \min(t_{A_1}, t_{A_2})$.
- Set $t_D = \min(S) = T_D$.

Loop:

- Case 1: $t_A \leq t_D$ and $t_A = t_{A_1}$ (Arrival of a class-1 customer)
 1. $t = t_A$.
 2. $T(i) = t$.
 3. $X(i) = X(i - 1) + 1$.
 4. $N_A = N_A + 1$.
 5. Update t_A : generate $t_{A_1} = t + Y_{A_1}$ and reset $t_A = \min(t_{A_1}, t_{A_2})$.
 6. Update t_D and Q_1 :
 - ✓ if $X(i) \leq N$,
 - (a) $Q_1 = 0$;
 - (b) generate a Y_{S_1} , replace one of the T_D s in vector S with $t + Y_{S_1}$ and reset $t_D = \min(S)$.
 - ✓ if $X(i) > N$
 - (a) $Q_1 = Q_1 + 1$;
 - (b) S and t_D remain the same.
- Case 2: $t_A \leq t_D$ and $t_A = t_{A_2}$ (Arrival of a class-2 customer)
 1. $t = t_A$.
 2. $T(i) = t$.
 3. $X(i) = X(i - 1) + 1$.

4. $N_A = N_A + 1$.
 5. Update t_A : generate $t_{A_2} = t + Y_{A_2}$ and reset $t_A = \min(t_{A_1}, t_{A_2})$.
 6. Update t_D and Q_2 :
 - ✓ if $X(i) \leq N$,
 - (a) $Q_2 = 0$;
 - (b) generate a Y_{S_2} , replace one of the T_D s in vector S with $t + Y_{S_2}$ and reset $t_D = \min(S)$.
 - ✓ if $X(i) > N$
 - (a) $Q_2 = Q_2 + 1$;
 - (b) S and t_D remain the same.
- Case 3: $t_A > t_D$ (Departure)
 1. $t = t_D$.
 2. $T(i) = t$.
 3. $X(i) = X(i - 1) - 1$.
 4. $N_D = N_D + 1$.
 5. Update t_D and Q_k ($k = 1, 2$):
 - ✓ if $X(i) < N$:
 - (a) $Q_k = 0$ ($k = 1, 2$);
 - (b) change the element t_D in S to T_D and reset $t_D = \min(S)$.
 - ✓ if $X(i) \geq N$:
 - * if $Q_1 > 0$,
 - (a) $Q_1 = Q_1 - 1$;
 - (b) generate a Y_{S_1} , replace the element t_D in S with $t + Y_{S_1}$ and reset $t_D = \min(S)$.
 - * if $Q_1 = 0$:
 - (a) $Q_2 = Q_2 - 1$;
 - (b) generate a Y_{S_2} , replace the element t_D in S with $t + Y_{S_2}$ and reset $t_D = \min(S)$.

A loop is terminated when $\min(t_A, t_D) > T_D$.

4.4.3 Simulation of N model

We simulate a N model in which pool 1 serves customers of class 1 and pool 2 serves customers of both classes (see Figure 4.5). The scheduling and routing policy considered here is keeping idleness in pool 2 and keeping waiting population in class 1. In other words, an arriving class-1 customer would choose to go to pool 1 if there are idle servers in both pools; and when a service in pool 2 ends, the idle server would give priority to class-2 customers if there are waiting customers in both classes.

Figure 4.5: N model

Let G_{A_k} denote the distribution of interarrival time of class- k customers ($k = 1, 2$). We assume that class- k customers are served by pool p , which consists of N_p statistically identical and independent servers, with service time distributed according to $G_{S_{kp}}$ distribution. Denote by Y_{A_1} , Y_{A_2} , $Y_{S_{11}}$, $Y_{S_{12}}$ and $Y_{S_{22}}$, respectively, the random variables with G_{A_1} , G_{A_2} , $G_{S_{11}}$, $G_{S_{12}}$ and $G_{S_{22}}$ distributions.

The following variables are introduced to do the simulation, where $p, k \in \{1, 2\}$.

- t : time variable
- T : $T(i)$ is the occurring time of the i th event
- X_p : $X_p(i)$ is the number of customers waiting to be or being served by pool p when i th event happens
- S_p : $S_p(j)$ is the service completion time of the j th server in pool p (at time t). It is a $N_p \times 1$ vector

- t_{A_k} : the next arrival time of class- k customers (after time t)
 t_{D_p} : the next departure time of customers served by pool- p servers (after time t)
 N_{A_k} : the number of arrivals of class- k customers (by time t)
 N_{D_p} : the number of departures from pool p (by time t)

Initialization:

- Set $t = 0$, $T(0) = 0$, $X_p(0) = 0$, $S_p = (T_D)_{N_p \times 1}$, $N_{A_k} = N_{D_p} = 0$. ($p \in \{1, 2\}$ and $k \in \{1, 2\}$)
- Generate $t_{A_k} = 0 + Y_{A_k}$. ($k \in \{1, 2\}$)
- Set $t_{D_p} = \min(S_p) = T_D$. ($p \in \{1, 2\}$)

Loop:

Take $t = \min(t_{A_1}, t_{A_2}, t_{D_1}, t_{D_2})$ and set $T(i) = t$.

- Case 1: $t = t_{A_1}$ (Arrival of a class-1 customer)
 1. $N_{A_1} = N_{A_1} + 1$.
 2. Update t_{A_1} : generate $t_{A_1} = t + Y_{A_1}$.
 3. Update X_p and t_{D_p} ($p = 1, 2$):
 - ✓ if $X_1(i-1) < N_1$
 - (a) $X_1(i) = X_1(i-1) + 1$; $X_2(i) = X_2(i-1)$.
 - (b) generate a $Y_{S_{11}}$, replace one of the T_D s in vector S_1 with $t + Y_{S_{11}}$ and reset $t_{D_1} = \min(S_1)$.
 - ✓ if $X_1(i-1) \geq N_1$ and $X_2(i-1) < N_2$
 - (a) $X_1(i) = X_1(i-1)$; $X_2(i) = X_2(i-1) + 1$.
 - (b) generate a $Y_{S_{12}}$, replace one of the T_D s in vector S_2 with $t + Y_{S_{12}}$ and reset $t_{D_2} = \min(S_2)$.
 - ✓ if $X_1(i-1) \geq N_1$ and $X_2(i-1) \geq N_2$
 - (a) $X_1(i) = X_1(i-1) + 1$; $X_2(i) = X_2(i-1)$.
- Case 2: $t = t_{A_2}$ (Arrival of a class-2 customer)
 1. $N_{A_2} = N_{A_2} + 1$.

2. Update t_{A_2} : generate $t_{A_2} = t + Y_{A_2}$.
 3. Update X_p : $X_1(i) = X_1(i - 1)$; $X_2(i) = X_2(i - 1) + 1$.
 4. Update t_{D_p} :
 - ✓ if $X_2(i - 1) < N_2$, generate a $Y_{S_{22}}$, replace one of the T_D s in vector S_2 with $t + Y_{S_{22}}$ and reset $t_{D_2} = \min(S_2)$.
 - ✓ if $X_2(i - 1) \geq N_2$, S_2 and t_{D_2} maintain the same.
- Case 3: $t = t_{D_1}$ (Departure from pool 1)
 1. $N_{D_1} = N_{D_1} + 1$.
 2. Update X_p : $X_1(i) = X_1(i - 1) - 1$; $X_2(i) = X_2(i - 1)$.
 3. Update t_{D_p} :
 - ✓ if $X_1(i - 1) \leq N_1$, replace the element t_{D_1} in S_1 with T_D and reset $t_{D_1} = \min(S_1)$.
 - ✓ if $X_1(i - 1) > N_1$, generate a $Y_{S_{11}}$, replace the element t_{D_1} in S_1 with $t + Y_{S_{11}}$ and reset $t_{D_1} = \min(S_1)$.
 - Case 4: $t = t_{D_2}$ (Departure from pool 2)
 1. $N_{D_2} = N_{D_2} + 1$.
 2. Update X_p and t_{D_p} :
 - ✓ if $X_2(i - 1) > N_2$
 - (a) $X_1(i) = X_1(i - 1)$; $X_2(i) = X_2(i - 1) - 1$.
 - (b) generate a $Y_{S_{22}}$, replace the element t_{D_2} in S_2 with $t + Y_{S_{22}}$ and reset $t_{D_2} = \min(S_2)$.
 - ✓ if $X_2(i - 1) \leq N_2$ and $X_1(i - 1) > N_1$
 - (a) $X_1(i) = X_1(i - 1) - 1$; $X_2(i) = X_2(i - 1)$.
 - (b) generate a $Y_{S_{12}}$, replace the element t_{D_2} in S_2 with $t + Y_{S_{12}}$ and reset $t_{D_2} = \min(S_2)$.
 - ✓ if $X_2(i - 1) \leq N_2$ and $X_1(i - 1) \leq N_1$
 - (a) $X_1(i) = X_1(i - 1)$; $X_2(i) = X_2(i - 1) - 1$.
 - (b) replace the element t_{D_2} in S_2 with T_D and reset $t_{D_2} = \min(S_2)$.

A loop is terminated when $\min(t_{A_1}, t_{A_2}, t_{D_1}, t_{D_2}) > T_D$.

Bibliography

- [1] D. Gross and C. M. Harris, *Fundamentals of queuing theory*. Wiley New York, 1998.
- [2] N. Gans, G. Koole, and A. Mandelbaum, “Telephone call centers: Tutorial, review, and research prospects,” *Manufacturing & Service Operations Management*, vol. 5, no. 2, pp. 79–141, 2003.
- [3] G. Shaikhet, M. M. Karbasioun, E. Kranakis, and I. Lambadaris, “Asymptotic convex optimization for packing random malleable demands in smart grid,” in *Communications (ICC), 2013 IEEE International Conference on*, pp. 2555–2560, IEEE, 2013.
- [4] X. Su and S. Zenios, “Patient choice in kidney allocation: The role of the queueing discipline,” *Manufacturing & Service Operations Management*, vol. 6, no. 4, pp. 280–301, 2004.
- [5] E. Almedhawe, B. Jewkes, and Q. He, “A markovian queueing model for ambulance offload delays,” *European Journal of Operational Research*, 2012.
- [6] A. S. Alfa and I. Frigui, “Discrete NT-policy single server queue with markovian arrival process and phase type service,” *European journal of operational research*, vol. 88, no. 3, pp. 599–613, 1996.
- [7] G. Brigham, “On a congestion problem in an aircraft factory,” *Operations Research*, vol. 3, no. 4, pp. 412–428, 1955.
- [8] O. Garnett and A. Mandelbaum, “An introduction to skills-based routing and its operational complexities,” *Teaching notes*, 2000.
- [9] J. A. Van Mieghem, “Dynamic scheduling with convex delay costs: The generalized $c-\mu$ rule,” *The Annals of Applied Probability*, pp. 809–833, 1995.
- [10] A. Mandelbaum and A. L. Stolyar, “Scheduling flexible servers with convex delay costs: Heavy-traffic optimality of the generalized $c\mu$ -rule,” *Operations Research*, vol. 52, no. 6, pp. 836–855, 2004.
- [11] S. Bell and R. Williams, “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*, vol. 11, no. 3, pp. 608–649, 2001.

- [12] S. L. Bell and R. J. Williams, "Dynamic scheduling of a parallel server system in heavy traffic with complete resource pooling: Asymptotic optimality of a threshold policy," *Electronic J. of Probability*, vol. 10, pp. 1044–1115, 2005.
- [13] J. M. Harrison and A. Zeevi, "Dynamic scheduling of a multiclass queue in the halfin-whitt heavy traffic regime," *Operations Research*, vol. 52, no. 2, pp. 243–257, 2004.
- [14] R. Atar, A. Mandelbaum, and M. I. Reiman, "Scheduling a multi class queue with many exponential servers: Asymptotic optimality in heavy traffic," *The Annals of Applied Probability*, vol. 14, no. 3, pp. 1084–1134, 2004.
- [15] T. Tezcan and J. Dai, "Dynamic control of n-systems with many servers: Asymptotic optimality of a static priority policy in heavy traffic," *Operations Research*, vol. 58, no. 1, pp. 94–110, 2010.
- [16] I. Gurvich and W. Whitt, "Service-level differentiation in many-server service systems: A solution based on fixed-queue-ratio routing," *Operations Research*, vol. 29, pp. 567–588, 2007.
- [17] R. Atar, A. Mandelbaum, and G. Shaikhet, "Simplified control problems for multiclass many-server queueing systems," *Mathematics of Operations Research*, vol. 34, no. 4, pp. 795–812, 2009.
- [18] G. Shaikhet, *Control of many-server queueing systems in heavy traffic*. PhD thesis, Technion, 2007. http://www.math.cmu.edu/~shaikhet/_docs/PhD_full.pdf.
- [19] M. Armony and A. R. Ward, "Fair dynamic routing in large-scale heterogeneous-server systems," *Operations Research*, vol. 58, no. 3, pp. 624–637, 2010.
- [20] T. Tezcan, "Asymptotically optimal control of many-server heterogeneous service systems with hyper-exponential service times," *Preprint*, 2007.
- [21] J. Dai and S. He, "Queues in service systems: customer abandonment and diffusion approximations," *Tutorials in Operations Research*, pp. 36–59, 2011.
- [22] S. M. Ross, *Simulation*. Elsevier Academic Press, 4th ed., 2006.
- [23] S. Halfin and W. Whitt, "Heavy-traffic limits for queues with many exponential servers," *Operations research*, vol. 29, no. 3, pp. 567–588, 1981.
- [24] O. Garnett, A. Mandelbaum, and M. Reiman, "Designing a call center with impatient customers," *Manufacturing & Service Operations Management*, vol. 4, no. 3, pp. 208–227, 2002.
- [25] J. M. Harrison and M. J. López, "Heavy traffic resource pooling in parallel-server systems," *Queueing systems*, vol. 33, no. 4, pp. 339–368, 1999.
- [26] R. Atar, "A diffusion model of scheduling control in queueing systems with many servers," *The Annals of Applied Probability*, vol. 15, no. 1B, pp. 820–852, 2005.
- [27] M. L. Puterman, *Markov decision processes: Discrete dynamic stochastic programming*. John Wiley & Sons, 1994.

- [28] X. Guo and O. Hernández-Lerma, *Continuous-time Markov decision processes: theory and applications*. New York: Springer, 2009.
- [29] R. Bellman, “Dynamic programming,” *Princeton University Press, 1957b. ISBN*, vol. 1050546524, p. 3, 1957.
- [30] D. P. Bertsekas and D. A. Castañon, “Adaptive aggregation methods for infinite horizon dynamic programming,” *Automatic Control, IEEE Transactions on*, vol. 34, no. 6, pp. 589–598, 1989.
- [31] R. A. Howard, “Dynamic programming and markov processes,” *Cambridge Mass.: MITP*, 1960.

Appendix A

Code for Markov Decision Process

A.1 Optimal control of V model

```
close all;
clear all;
alpha=0.9; % Discount factor.
lambda_1=24; % The arrival rate of class-1 customers.
lambda_2=24; % The arrival rate of class-2 customers.
mu_1=6; % The service rate of class-1 customers.
mu_2=6; % The service rate of class-2 customers.
theta_1=5; % The abandonment rate of class-1 customers.
theta_2=1; % The abandonment rate of class-2 customers.
N=8; % The number of servers.
X_1=20; % The boundary of number of class-1 customers.
X_2=20; % The boundary of number of class-2 customers.
c_1=10; % Cost of a waiting class-1 customer.
c_2=5; % Cost of a waiting class-2 customer.

A=cell(1,(X_1+1)*(X_2+1)); % The array of action sets for all states.
F=zeros((X_1+1)*(X_2+1),100); % Order of policy
F(:,1)=ones((X_1+1)*(X_2+1),1); % Order of initial policy
Q=zeros((X_1+1)*(X_2+1),(X_1+1)*(X_2+1));
C=zeros((X_1+1)*(X_2+1),1); % Cost vector.
SA=zeros((X_1+1)*(X_2+1),4); % Used to save states and optimal policy.
for x_1=0:X_1 % x_1 is the number of class-1 customers
```

```

for x_2=0:X_2 % x_2 is the number of class-2 customers
    % Find the corresponding i for (x_1,x_2).
    i=transform(x_1,x_2,X_1,X_2);
    % transform is a function defined in transform.m
    SA(i,1:2)=[x_1,x_2];
    SA(i,5)=x_1+x_2;
    % Define the action set for this state.
    if x_1+x_2<=N
        A{i}=[x_1,x_2];
    else
        if x_1<N
            if x_2<N
                for k=1:(x_1+x_2-N+1)
                    A{i}(k,:)=[x_1-k+1,N-x_1+k-1];
                end
            else
                for k=1:(x_1+1)
                    A{i}(k,:)=[k-1,N-k+1];
                end
            end
        else
            if x_2<N
                for k=1:(x_2+1)
                    A{i}(k,:)=[N-k+1,k-1];
                end
            else
                for k=1:(N+1)
                    A{i}(k,:)=[k-1,N-k+1];
                end
            end
        end
    end
end
% Create the transition rates matrix with initial policy.
if x_1<X_1

```

```

        j=transform(x_1+1,x_2,X_1,X_2);
        Q(i,j)=lambda_1;
    end
    if x_2<X_2
        j=transform(x_1,x_2+1,X_1,X_2);
        Q(i,j)=lambda_2;
    end
    if x_1>0
        j=transform(x_1-1,x_2,X_1,X_2);
        Q(i,j)=(mu_1-theta_1)*A{i}(F(i,1),1)+x_1*theta_1;
        % Pick the F(i)th policy for state i.
    end
    if x_2>0
        j=transform(x_1,x_2-1,X_1,X_2);
        Q(i,j)=(mu_2-theta_2)*A{i}(F(i,1),2)+x_2*theta_2;
    end
    Q(i,i)=-sum(Q(i,:));
    C(i)=cost(x_1,A{i}(F(i,1),1),x_2,A{i}(F(i,1),2),c_1,c_2);
end
end
J=(alpha.*eye((X_1+1)*(X_2+1))-Q)\C;
% Evaluation of the initial policy.

for n=2:100
    clear Q;
    Q=zeros((X_1+1)*(X_2+1),(X_1+1)*(X_2+1));
    % Find the improved policy F(:,n)
    for x_1=0:X_1
        for x_2=0:X_2
            i=transform(x_1,x_2,X_1,X_2);
            % Try all the available actions for state i.
            for k=1:size(A{i},1)
                clear Q(i,:);
                Q(i,:)=zeros(1,(X_1+1)*(X_2+1));
            end
        end
    end
end

```

```

        if x_1<X_1
            j=transform(x_1+1,x_2,X_1,X_2);
            Q(i,j)=lambda_1;
        end
        if x_2<X_2
            j=transform(x_1,x_2+1,X_1,X_2);
            Q(i,j)=lambda_2;
        end
        if x_1>0
            j=transform(x_1-1,x_2,X_1,X_2);
            Q(i,j)=(mu_1-theta_1)*A{i}(k,1)+x_1*theta_1;
        end
        if x_2>0
            j=transform(x_1,x_2-1,X_1,X_2);
            Q(i,j)=(mu_2-theta_2)*A{i}(k,2)+x_2*theta_2;
        end
        Q(i,i)=-sum(Q(i,:));
        D=cost(x_1,A{i}(k,1),x_2,A{i}(k,2),c_1,c_2)+Q(i,:)*J;
        if round((D-alpha*J(i))*10^5)/10^5<0
            F(i,n)=k;
        end
    end
    if F(i,n)==0
        F(i,n)=F(i,n-1);
    end
end
end
P=F(:,n)-F(:,n-1);
id=find(P==0);
if length(id)==(X_1+1)*(X_2+1)
    break;
end
% Evaluate the improved policy
clear Q;

```

```

clear C;
Q=zeros((X_1+1)*(X_2+1),(X_1+1)*(X_2+1));
C=zeros((X_1+1)*(X_2+1),1);
for x_1=0:X_1
    for x_2=0:X_2
        i=transform(x_1,x_2,X_1,X_2);
        if x_1<X_1
            j=transform(x_1+1,x_2,X_1,X_2);
            Q(i,j)=lambda_1;
        end
        if x_2<X_2
            j=transform(x_1,x_2+1,X_1,X_2);
            Q(i,j)=lambda_2;
        end
        if x_1>0
            j=transform(x_1-1,x_2,X_1,X_2);
            Q(i,j)=(mu_1-theta_1)*A{i}(F(i,n),1)+x_1*theta_1;
        end
        if x_2>0
            j=transform(x_1,x_2-1,X_1,X_2);
            Q(i,j)=(mu_2-theta_2)*A{i}(F(i,n),2)+x_2*theta_2;
        end
        Q(i,i)=-sum(Q(i,:));
        C(i)=cost(x_1,A{i}(F(i,n),1),x_2,A{i}(F(i,n),2),c_1,c_2);
    end
end
clear J;
J=(alpha.*eye((X_1+1)*(X_2+1))-Q)\C;
end

Optimal=F(:,n);

for k=1:(X_1+1)*(X_2+1)
    SA(k,3:4)=A{k}(F(k,n),:);

```

```

end

subplot(1,2,1);
for w=1:(X_1+1)*(X_2+1)
    hold on;
    if SA(w,3)==(min(SA(w,1),N))
        if SA(w,4)==(min(SA(w,2),N))
            plot3(SA(w,1),SA(w,2),J(w),'ro','color','b','Marker','+');
        else % Priority is given to class 1
            plot3(SA(w,1),SA(w,2),J(w),'ro','color','m','Marker','o');
        end
    else % Priority is given to class 2
        if SA(w,4)==(min(SA(w,2),N))
            plot3(SA(w,1),SA(w,2),J(w),'ro','color','r','Marker','*');
        end
    end
end
end

E=cell(1,2*(N+ceil(sqrt(N)))-1);
% Used to collect the states will be under consideration
G=zeros(2*(N+ceil(sqrt(N)))-1,2);
% Used to collect the minimal value for each "sum".
subplot(1,2,2);
for x_1=1:(N+ceil(sqrt(N)))
    for x_2=1:(N+ceil(sqrt(N)))
        i=transform(x_1,x_2,X_1,X_2);
        E{SA(i,5)-1}=[E{SA(i,5)-1};i];
        % Collect the states that have the same sum.
        hold on;
        if SA(i,3)==(min(SA(i,1),N))
            if SA(i,4)==(min(SA(i,2),N))
                plot(SA(i,5),J(i),'color','b','Marker','+');
            else
                plot(SA(i,5),J(i),'color','m','Marker','o');
            end
        end
    end
end

```

```

        end
    else
        if SA(i,4)==(min(SA(i,2),N))
            plot(SA(i,5),J(i),'color','r','Marker','*');
        end
    end
end
end

for k=1:(2*(N+ceil(sqrt(N)))-1)
    G(k,:)=[(k+1),min(J(E{k}))];
end

Slope=diff(G(:,2))./diff(G(:,1));

% Cost function
function C=cost(x,s,y,t,p,q)
C=p*(x-s)+q*(y-t);
end

% Define a function to transform two-dimensional states (x_1,x_2) to
% one-dimensional state i.
function i=transform(a,b,A,B)
Min=min(A,B);
Max=max(A,B);
Sum=a+b;
if Sum==0
    i=1;
else
    if Sum<=Min
        i=(1+Sum)*Sum/2+a+1;
    else
        if Sum<Max
            i=(1+Min)*Min/2+(Min+1)*(Sum-Min)+min(Sum,B)-b+1;

```

```

        else
            i=(1+Min)*Min/2+(Min+1)*(Max-Min)+(A+B-Sum+Min+3)
            *(Min-(A+B)+Sum)/2+(B-b)+1;
        end
    end
end
end
end

```

A.2 Optimal control of N model

```

close all;
clear all;
alpha=0.9; % Discount factor.
lambda_1=500; % The arrival rate of class-1 customers.
lambda_2=200; % The arrival rate of class-2 customers.
mu_1=6; % The service rate of customers served by pool 1.
mu_2=8; % The service rate of customers served by pool 2.
theta_1=5; % The abandonment rate of class-1 customers.
theta_2=1; % The abandonment rate of class-2 customers.
N_1=50; % The number of servers in pool 1.
N_2=50; % The number of servers in pool 2.
X_1=N_1+N_2; % The boundary of number of class-1 customers.
X_2=N_2; % The boundary of number of class-2 customers.
c_1=8; % Cost of a waiting class-1 customer.
c_2=5; % Cost of a waiting class-2 customer.

A=cell(1,(X_1+1)*(X_2+1)); % The array of action sets for all states.
F=zeros((X_1+1)*(X_2+1),100); % Order of policy
F(:,1)=ones((X_1+1)*(X_2+1),1); % Order of initial policy
Q=zeros((X_1+1)*(X_2+1),(X_1+1)*(X_2+1));
C=zeros((X_1+1)*(X_2+1),1); % Cost vector.
SA=zeros((X_1+1)*(X_2+1),6); % Used to save states and optimal policy.
for x_1=0:X_1 % x_1 is the number of class-1 customers.
    for x_2=0:X_2 % x_2 is the number of class-2 customers.

```



```

% Find the corresponding i for (x_1,x_2).
i=transform(x_1,x_2,X_1,X_2);
% transform is a function defined in transform.m
SA(i,1:2)=[x_1,x_2];
SA(i,6)=x_1+x_2;
% Define the action set for this state.
if (x_1+x_2)<=(N_1+N_2)
    m=min((N_2-x_2),x_1)-max((x_1-N_1),0)+1;
    for k=1:m
        s_22=x_2;
        % s_22 is number of class-2 customers served by pool 2.
        s_12=min((N_2-x_2),x_1)-k+1;
        % s_21 is number of class-1 customers served by pool 2.
        s_11=x_1-s_12;
        % s_11 is number of class-1 customers served by pool 1.
        A{i}(k,:)= [s_11,s_12,s_22];
    end
else
    m=min((x_1-N_1),N_2)-(N_2-x_2)+1;
    for k=1:m
        s_11=N_1;
        s_12=min((x_1-N_1),N_2)-k+1;
        s_22=N_2-s_12;
        A{i}(k,:)= [s_11,s_12,s_22];
    end
end
% Create the transition rates matrix with initial policy.
if x_1<X_1
    j=transform(x_1+1,x_2,X_1,X_2);
    Q(i,j)=lambda_1;
end
if x_2<X_2
    j=transform(x_1,x_2+1,X_1,X_2);
    Q(i,j)=lambda_2;
end

```

```

end
if x_1>0
    j=transform(x_1-1,x_2,X_1,X_2);
    Q(i,j)=theta_1*(x_1-A{i}(F(i,1),1)-A{i}(F(i,1),2))
        +mu_1*A{i}(F(i,1),1)+mu_2*A{i}(F(i,1),2);
    % Pick the F(i)th policy for state i.
end
if x_2>0
    j=transform(x_1,x_2-1,X_1,X_2);
    Q(i,j)=theta_2*(x_2-A{i}(F(i,1),3))+mu_2*A{i}(F(i,1),3);
end
Q(i,i)=-sum(Q(i,:));
C(i)=cost(x_1,x_2,A{i}(F(i,1),1),A{i}(F(i,1),2),
    A{i}(F(i,1),3),c_1,c_2);
end
end
J=(alpha.*eye((X_1+1)*(X_2+1))-Q)\C;
% Evaluation of the initial policy.

for n=2:100
    clear Q;
    Q=zeros((X_1+1)*(X_2+1),(X_1+1)*(X_2+1));
    % Find the improved policy F(:,n)
    for x_1=0:X_1
        for x_2=0:X_2
            i=transform(x_1,x_2,X_1,X_2);
            % Try all the available actions for state i.
            for k=1:size(A{i},1)
                clear Q(i,:);
                Q(i,:)=zeros(1,(X_1+1)*(X_2+1));
                if x_1<X_1
                    j=transform(x_1+1,x_2,X_1,X_2);
                    Q(i,j)=lambda_1;
                end
            end
        end
    end
end

```

```

        if x_2<X_2
            j=transform(x_1,x_2+1,X_1,X_2);
            Q(i,j)=lambda_2;
        end
        if x_1>0
            j=transform(x_1-1,x_2,X_1,X_2);
            Q(i,j)=theta_1*(x_1-A{i}(k,1)-A{i}(k,2))
                +mu_1*A{i}(k,1)+mu_2*A{i}(k,2);
        end
        if x_2>0
            j=transform(x_1,x_2-1,X_1,X_2);
            Q(i,j)=theta_2*(x_2-A{i}(k,3))+mu_2*A{i}(k,3);
        end
        Q(i,i)=-sum(Q(i,:));
        D=cost(x_1,x_2,A{i}(k,1),A{i}(k,2),A{i}(k,3),c_1,c_2)
            +Q(i,:)*J;
        if round((D-alpha*J(i))*10^5)/10^5<0
            F(i,n)=k;
        end
    end
    if F(i,n)==0
        F(i,n)=F(i,n-1);
    end
end
end
P=F(:,n)-F(:,n-1);
id=find(P==0);
if length(id)==(X_1+1)*(X_2+1)
    break;
end
% Evaluate the improved policy
clear Q;
clear C;
Q=zeros((X_1+1)*(X_2+1),(X_1+1)*(X_2+1));

```

```

C=zeros((X_1+1)*(X_2+1),1);
for x_1=0:X_1
    for x_2=0:X_2
        i=transform(x_1,x_2,X_1,X_2);
        if x_1<X_1
            j=transform(x_1+1,x_2,X_1,X_2);
            Q(i,j)=lambda_1;
        end
        if x_2<X_2
            j=transform(x_1,x_2+1,X_1,X_2);
            Q(i,j)=lambda_2;
        end
        if x_1>0
            j=transform(x_1-1,x_2,X_1,X_2);
            Q(i,j)=theta_1*(x_1-A{i}(F(i,n),1)-A{i}(F(i,n),2))
                +mu_1*A{i}(F(i,n),1)+mu_2*A{i}(F(i,n),2);
        end
        if x_2>0
            j=transform(x_1,x_2-1,X_1,X_2);
            Q(i,j)=theta_2*(x_2-A{i}(F(i,n),3))
                +mu_2*A{i}(F(i,n),3);
        end
        Q(i,i)=-sum(Q(i,:));
        C(i)=cost(x_1,x_2,A{i}(F(i,n),1),A{i}(F(i,n),2),
            A{i}(F(i,n),3),c_1,c_2);
    end
end
clear J;
J=(alpha.*eye((X_1+1)*(X_2+1))-Q)\C;
end

Optimal=F(:,n);

for k=1:(X_1+1)*(X_2+1)

```

```

    SA(k,3:5)=A{k}(F(k,n),:);
end

subplot(1,2,1);
for w=1:(X_1+1)*(X_2+1)
    hold on;
    if SA(w,6)<=(N_1+N_2)
        if SA(w,3)==max((SA(w,6)-N_2),0) % Keep idleness in pool 1
            plot3(SA(w,1),SA(w,2),J(w),'ro','color','g','Marker','x');
        else
            if SA(w,3)==min(SA(w,1),N_1) % Keep idleness in pool 2
                plot3(SA(w,1),SA(w,2),J(w),'ro','color','k','Marker','x');
            else
                plot3(SA(w,1),SA(w,2),J(w),'ro','color','b','Marker','+');
            end
        end
    end
else
    if SA(w,4)==min((SA(w,1)-N_1),N_2)
        % Priority is given to class 1 for pool 2
        plot3(SA(w,1),SA(w,2),J(w),'ro','color','m','Marker','o');
    else
        if SA(w,4)==N_2-SA(w,2)
            % Priority is given to class 2 for pool 2
            plot3(SA(w,1),SA(w,2),J(w),'ro','color','r','Marker','*');
        else
            plot3(SA(w,1),SA(w,2),J(w),'ro','color','b','Marker','+');
        end
    end
end
end

N=N_1+N_2;
E=cell(1,N+ceil(2*N^(2/3)));
% Used to collect the states will be under consideration

```

```

E{1}=0;
G=zeros(N+ceil(2*N^(2/3))-1,2);
% Used to collect the minimal value for each "sum".
subplot(1,2,2);
for x_1=1:X_1
    for x_2=1:X_2
        i=transform(x_1,x_2,X_1,X_2);
        if SA(i,6)<=N+ceil(2*N^(2/3))
            E{SA(i,6)}=[E{SA(i,6)};i];
            % Collect the states that have the same sum.
            hold on;
            if SA(i,6)<=(N_1+N_2)
                if SA(i,3)==max((SA(i,6)-N_2),0)
                    plot(SA(i,6),J(i),'color','g','Marker','x');
                else
                    if SA(i,3)==min(SA(i,1),N_1)
                        plot(SA(i,6),J(i),'color','k','Marker','x');
                    else
                        plot(SA(i,6),J(i),'color','b','Marker','+');
                    end
                end
            end
        else
            if SA(i,4)==min((SA(i,1)-N_1),N_2)
                plot(SA(i,6),J(i),'color','m','Marker','o');
            else
                if SA(i,4)==N_2-SA(i,2)
                    plot(SA(i,6),J(i),'color','r','Marker','*');
                else
                    plot(SA(i,6),J(i),'color','b','Marker','+');
                end
            end
        end
    end
end
end
end

```

```

end

for k=1:(N+ceil(2*N^(2/3))-1)
    G(k,:)=[k+1,min(J(E{k+1}))];
end
Slope=diff(G(:,2))./diff(G(:,1));

% Cost function
function C=cost(x,y,s1,s2,s3,p,q)
C=p*(x-s1-s2)+q*(y-s3);
end

% Define a function to transform two-dimensional states (x_1,x_2) to
% one-dimensional state i.
function i=transform(a,b,A,B)
if A<B
    Min=A;
    Max=B;
else
    if A>B
        Min=B;
        Max=A;
    else
        Min=A;
        Max=A;
    end
end
Sum=a+b;
if Sum==0
    i=1;
else
    if Sum<=Min
        i=(1+Sum)*Sum/2+a+1;
    else

```

```

        if Sum<Max
            i=(1+Min)*Min/2+(Min+1)*(Sum-Min)+min(Sum,B)-b+1;
        else
            i=(1+Min)*Min/2+(Min+1)*(Max-Min)+(A+B-Sum+Min+3)
              *(Min-(A+B)+Sum)/2+(B-b)+1;
        end
    end
end
end
end

```

A.3 Optimal control of X model

```

close all;
clear all;
alpha=0.9; % Discount factor.
lambda_1=140; % The arrival rate of class-1 customers.
lambda_2=140; % The arrival rate of class-2 customers.
mu_1=6; % The service rate of customers served by pool 1.
mu_2=8; % The service rate of customers served by pool 2.
theta_1=5; % The abandonment rate of class-1 customers.
theta_2=1; % The abandonment rate of class-2 customers.
N_1=20; % The number of servers in pool 1.
N_2=20; % The number of servers in pool 2.
X_1=70; % The boundary of number of class-1 customers.
X_2=70; % The boundary of number of class-2 customers.
c_1=12; % Cost of a waiting class-1 customer.
c_2=5; % Cost of a waiting class-2 customer.

A=cell(1,(X_1+1)*(X_2+1)); % The array of action sets for all states.
F=zeros((X_1+1)*(X_2+1),100); % Order of policy
F(:,1)=ones((X_1+1)*(X_2+1),1); % Order of initial policy
Q=zeros((X_1+1)*(X_2+1),(X_1+1)*(X_2+1));
C=zeros((X_1+1)*(X_2+1),1); % Cost vector.

```



```

SA=zeros((X_1+1)*(X_2+1),7); % Used to save states and optimal policy.
for x_1=0:X_1 % x_1 is the number of class-1 customers.
    for x_2=0:X_2 % x_2 is the number of class-2 customers.
        % Find the corresponding i for (x_1,x_2).
        i=transform(x_1,x_2,X_1,X_2);
        % transform is a function defined in transform.m
        SA(i,1:2)=[x_1,x_2];
        SA(i,7)=x_1+x_2;
        % Define the action set for this state.
        if x_1+x_2<=(N_1+N_2)
            clear m_2;
            m_1=min(x_1,N_1)-max((x_1-N_2),0)+1;
            m_2=zeros(m_1,1);
            for k=1:m_1
                s_11=min(x_1,N_1)-k+1;
                % s_11 is number of class-1 customers served by pool 1.
                s_12=x_1-s_11;
                % s_12 is number of class-1 customers served by pool 2.
                m_2(k)=min((N_1-s_11),x_2)-max((x_1+x_2-N_2-s_11),0)+1;
                for r=1:m_2(k)
                    s_21=min((N_1-s_11),x_2)-r+1;
                    % s_21 is number of class-2 customers served by pool 1.
                    s_22=x_2-s_21;
                    % s_22 is number of class-2 customers served by pool 2.
                    rr=0;
                    if k==1
                        ord=r;
                    else
                        for kk=1:(k-1)
                            rr=rr+m_2(kk);
                        end
                        ord=rr+r;
                    end
                    A{i}(ord,:)= [s_11,s_21,s_12,s_22];
                end
            end
        end
    end
end

```

```

        end
    end
else
    clear m_2;
    m_1=min(x_1,N_1)-max((N_1-x_2),0)+1;
    m_2=zeros(m_1,1);
    for k=1:m_1
        s_11=min(x_1,N_1)-k+1;
        s_21=N_1-s_11;
        m_2(k)=min((x_1-s_11),N_2)-max((N_1+N_2-x_2-s_11),0)+1;
        for r=1:m_2(k)
            s_12=min((x_1-s_11),N_2)-r+1;
            s_22=N_2-s_12;
            rr=0;
            if k==1
                ord=r;
            else
                for kk=1:(k-1)
                    rr=rr+m_2(kk);
                end
                ord=rr+r;
            end
            A{i}(ord,:)= [s_11,s_21,s_12,s_22];
        end
    end
end

end

% Create the transition rates matrix with initial policy.
if x_1<X_1
    j=transform(x_1+1,x_2,X_1,X_2);
    Q(i,j)=lambda_1;
end
if x_2<X_2
    j=transform(x_1,x_2+1,X_1,X_2);
    Q(i,j)=lambda_2;
end

```

```

end
if x_1>0
    j=transform(x_1-1,x_2,X_1,X_2);
    Q(i,j)=theta_1*(x_1-A{i}(F(i,1),1)-A{i}(F(i,1),3))
        +mu_1*A{i}(F(i,1),1)+mu_2*A{i}(F(i,1),3);
    % Pick the F(i)th policy for state i.
end
if x_2>0
    j=transform(x_1,x_2-1,X_1,X_2);
    Q(i,j)=theta_2*(x_2-A{i}(F(i,1),2)-A{i}(F(i,1),4))
        +mu_1*A{i}(F(i,1),2)+mu_2*A{i}(F(i,1),4);
end
Q(i,i)=-sum(Q(i,:));
C(i)=cost(x_1,x_2,A{i}(F(i,1),1),A{i}(F(i,1),2),
    A{i}(F(i,1),3),A{i}(F(i,1),4),c_1,c_2);
end
end
J=(alpha.*eye((X_1+1)*(X_2+1))-Q)\C;
% Evaluation of the initial policy.

for n=2:100
    clear Q;
    Q=zeros((X_1+1)*(X_2+1),(X_1+1)*(X_2+1));
    % Find the improved policy F(:,n)
    for x_1=0:X_1
        for x_2=0:X_2
            i=transform(x_1,x_2,X_1,X_2);
            % Try all the available actions for state i.
            for k=1:size(A{i},1)
                clear Q(i,:);
                Q(i,:)=zeros(1,(X_1+1)*(X_2+1));
                if x_1<X_1
                    j=transform(x_1+1,x_2,X_1,X_2);
                    Q(i,j)=lambda_1;

```

```

end
if x_2<X_2
    j=transform(x_1,x_2+1,X_1,X_2);
    Q(i,j)=lambda_2;
end
if x_1>0
    j=transform(x_1-1,x_2,X_1,X_2);
    Q(i,j)=theta_1*(x_1-A{i}(k,1)-A{i}(k,3))
        +mu_1*A{i}(k,1)+mu_2*A{i}(k,3);
end
if x_2>0
    j=transform(x_1,x_2-1,X_1,X_2);
    Q(i,j)=theta_2*(x_2-A{i}(k,2)-A{i}(k,4))
        +mu_1*A{i}(k,2)+mu_2*A{i}(k,4);
end
Q(i,i)=-sum(Q(i,:));
D=cost(x_1,x_2,A{i}(k,1),A{i}(k,2),A{i}(k,3),
    A{i}(k,4),c_1,c_2)+Q(i,:)*J;
if round((D-alpha*J(i))*10^5)/10^5<0
    F(i,n)=k;
end
end
if F(i,n)==0
    F(i,n)=F(i,n-1);
end
end
end
P=F(:,n)-F(:,n-1);
id=find(P==0);
if length(id)==(X_1+1)*(X_2+1)
    break;
end
% Evaluate the improved policy
clear Q;

```

```

clear C;
Q=zeros((X_1+1)*(X_2+1),(X_1+1)*(X_2+1));
C=zeros((X_1+1)*(X_2+1),1);
for x_1=0:X_1
    for x_2=0:X_2
        i=transform(x_1,x_2,X_1,X_2);
        if x_1<X_1
            j=transform(x_1+1,x_2,X_1,X_2);
            Q(i,j)=lambda_1;
        end
        if x_2<X_2
            j=transform(x_1,x_2+1,X_1,X_2);
            Q(i,j)=lambda_2;
        end
        if x_1>0
            j=transform(x_1-1,x_2,X_1,X_2);
            Q(i,j)=theta_1*(x_1-A{i}(F(i,n),1)-A{i}(F(i,n),3))
                +mu_1*A{i}(F(i,n),1)+mu_2*A{i}(F(i,n),3);
        end
        if x_2>0
            j=transform(x_1,x_2-1,X_1,X_2);
            Q(i,j)=theta_2*(x_2-A{i}(F(i,n),2)-A{i}(F(i,n),4))
                +mu_1*A{i}(F(i,n),2)+mu_2*A{i}(F(i,n),4);
        end
        Q(i,i)=-sum(Q(i,:));
        C(i)=cost(x_1,x_2,A{i}(F(i,n),1),A{i}(F(i,n),2),
            A{i}(F(i,n),3),A{i}(F(i,n),4),c_1,c_2);
    end
end
clear J;
J=(alpha.*eye((X_1+1)*(X_2+1))-Q)\C;
end

Optimal=F(:,n);

```

```

for k=1:(X_1+1)*(X_2+1)
    SA(k,3:6)=A{k}(F(k,n),:);
end

subplot(1,2,1);
for w=1:(X_1+1)*(X_2+1)
    hold on;
    if SA(w,7)<=(N_1+N_2)
        if (SA(w,3)+SA(w,4))==max((SA(w,7)-N_2),0)
            % Keep idleness in pool 1
            plot3(SA(w,1),SA(w,2),J(w),'ro','color','g','Marker','x');
        else
            if (SA(w,3)+SA(w,4))==min(SA(w,7),N_1)
                % Keep idleness in pool 2
                plot3(SA(w,1),SA(w,2),J(w),'ro','color','k','Marker','x');
            else
                plot3(SA(w,1),SA(w,2),J(w),'ro','color','b','Marker','+');
            end
        end
    end
else
    if (SA(w,3)+SA(w,5))==min(SA(w,1),(N_1+N_2))
        % Priority is given to class 1
        plot3(SA(w,1),SA(w,2),J(w),'ro','color','m','Marker','o');
    else
        if (SA(w,3)+SA(w,5))==max((N_1+N_2)-SA(w,2),0)
            % Priority is given to class 2
            plot3(SA(w,1),SA(w,2),J(w),'ro','color','r','Marker','*');
        else
            plot3(SA(w,1),SA(w,2),J(w),'ro','color','b','Marker','+');
        end
    end
end
end
end

```

```

N=N_1+N_2;
E=cell(1,N+2*ceil(N_1^(2/3)+N_2^(2/3)));
% Used to collect the states will be under consideration
E{1}=0;
G=zeros(N+2*ceil(N_1^(2/3)+N_2^(2/3))-1,2);
% Used to collect the minimal value for each "sum".
subplot(1,2,2);
for x_1=1:X_1
    for x_2=1:X_2
        if (x_1+x_2)<=(N+2*ceil(N_1^(2/3)+N_2^(2/3)))
            i=transform(x_1,x_2,X_1,X_2);
            E{SA(i,7)}=[E{SA(i,7)};i];
            % Collect the states that have the same sum.
            hold on;
            if SA(i,7)<=(N_1+N_2)
                if (SA(i,3)+SA(i,4))==max((SA(i,7)-N_2),0)
                    plot(SA(i,7),J(i),'color','g','Marker','x');
                else
                    if (SA(i,3)+SA(i,4))==min(SA(i,7),N_1)
                        plot(SA(i,7),J(i),'color','k','Marker','x');
                    else
                        plot(SA(i,7),J(i),'color','b','Marker','+');
                    end
                end
            end
        else
            if (SA(i,3)+SA(i,5))==min(SA(i,1),(N_1+N_2))
                plot(SA(i,7),J(i),'color','m','Marker','o');
            else
                if (SA(i,3)+SA(i,5))==max((N_1+N_2)-SA(i,2),0)
                    plot(SA(i,7),J(i),'color','r','Marker','*');
                else
                    plot(SA(i,7),J(i),'color','b','Marker','+');
                end
            end
        end
    end
end

```

```

                                end
                            end
                        end
                    end
                end

for k=1:(N+2*ceil(N_1^(2/3)+N_2^(2/3))-1)
    G(k,:)=[k+1,min(J(E{k+1}))];
end

Slope=diff(G(:,2))./diff(G(:,1));

% Cost function
function C=cost(x,y,s1,s2,s3,s4,p,q)
C=p*(x-s1-s3)+q*(y-s2-s4);
end

% Define a function to transform two-dimensional states (x_1,x_2) to
% one-dimensional state i.
function i=transform(a,b,A,B)
if A<B
    Min=A;
    Max=B;
else
    if A>B
        Min=B;
        Max=A;
    else
        Min=A;
        Max=A;
    end
end
end
Sum=a+b;
if Sum==0

```



```

        i=1;
    else
        if Sum<=Min
            i=(1+Sum)*Sum/2+a+1;
        else
            if Sum<Max
                i=(1+Min)*Min/2+(Min+1)*(Sum-Min)+min(Sum,B)-b+1;
            else
                i=(1+Min)*Min/2+(Min+1)*(Max-Min)+(A+B-Sum+Min+3)
                  *(Min-(A+B)+Sum)/2+(B-b)+1;
            end
        end
    end
end
end

```

Appendix B

Code for Discrete-Event Simulation

B.1 Simulation of G/G/N model

```
%=====
% We take a M/M/N queueing model as an example to check the feasibi-
% lity of this algorithm. We assume that this is an initially empty %
% system. Customers arrive according to Poisson process with rate 8 %
% and are served with exponentially distributed time of mean 1. The-
% re are 10 servers in total. The observation time is [0,1000].      %
%=====

close all;
clear all;
t=0;
T(1)=0;
X(1)=0; % X is number of people in the system
N=10;
N_A=0;
S=1000*ones(N,1);
% S is a vector used to save the service completion time.

t_A=t+exprnd(1/8); % t_A is next arrival time.
t_D=min(S); % t_D is next departure time.

for i=2:20000
```

```

if t_A<=t_D % upcoming event is arrival.
    t=t_A; % update time
    T(i)=t;
    X(i)=X(i-1)+1;
    N_A=N_A+1;
    if X(i)<=N % there are some free servers/there is no queue.
        F=find(S==1000); % F gives the index of free servers.
        ind=F(1);
        % assign the first free server to the upcoming customer
        S(ind)=t+exprnd(1); % departure time of the new customer
    end % if all the servers are busy,new customers will directly
        % join the queue,so vector S does not need to be updated.
    t_D=min(S); % time of next departure
    t_A=t+exprnd(1/8); % time of next arrival
    if t_A>1000
        break;
    end
else % upcoming event is departure.
    t=t_D;
    T(i)=t;
    X(i)=X(i-1)-1;
    if X(i)==0 % if there is no one in the system, next event will
        t_D=1000; % be arrival. t_D=1000 ensures that t_A<=t_D.
        S=1000*ones(N,1);
        % the system is empty,so the S should be initialized.
    else
        id=find(S==t_D); % find the departure one
        if X(i)>=N % someone leaves, someone
            S(id)=t+exprnd(1);% in the queue will be served.
        else
            S(id)=1000;
        end
    end
end
t_D=min(S);

```

```

        if t_D>1000
            break;
        end
    end
end
end
plot(T,X,'Color','blue');

Q=max(X-N,0); % length of queue
for k=1:(i-1)
    inter_T(k)=T(k+1)-T(k);
end
% average of X
ave_X=sum(inter_T.*X(1:(i-1)))/T(i)
% average length of queue L_q
L_Q=sum(inter_T.*Q(1:(i-1)))/T(i)
% average waiting time in queue W_q
W_Q=sum(inter_T.*Q(1:(i-1)))/N_A
% W=total waiting time/number of arrivals

```

B.2 Simulation of G/G/N with abandonment

```

%=====
% We take a M/M/N+M queueing model as an example to check the feasi-
% bility of this algorithm. Assume that this is an initially empty
% system. Customers arrive according to a Poisson process with rate
% 8 and they are served with exponentially distributed time of mean
% 1. Customer patience is modeled by exponentially distributed rand-
% om variables with mean 5. There are 10 servers in total. The obse-
% rvation time is [0,1000].
%=====

close all;
clear all;
t=0;

```

```

T(1)=0; % T is the occurring time of events.
X(1)=0; % X is the number of customers in the system.
N=10; % N is the number of servers.
N_A=0; % N_A is the number of arrival.
S=1000*ones(N,1);
% S is a vector used to save the service completion time.

t_A=t+exprnd(1/8);
t_D=1000;

for i=2:20000
    if t_A<=t_D % upcoming event is arrival.
        t=t_A;
        T(i)=t;
        X(i)=X(i-1)+1;
        N_A=N_A+1;
        if X(i)<=N % There is no queue. The newcomer will be served
            % directly. It's impossible for him to abandon.
            R(N_A)=1000; % R is a vector used to save the patience
                % time of every arrival.
            F=find(S==1000); % F gives the index of idle servers.
            ind=F(1);
            % assign the first idle server to the new arrival.
            S(ind)=t+exprnd(1);
            % service completion time of the new arrival.
        else % The new comer will join the queue in this situation.
            R(N_A)=t+exprnd(5); % patience time of the new customer.
        end
        t_D=min(min(S),min(R)); % time of next departure.
        if sum(ismember(S,t_D))==1 % 'ismember' check whether t_D
            % is an element of S.
            M=0; % It means t_D is an element of S. Someone will
                % complete service and leave.
        else

```

```

        M=1;
        % It means t_D is an element of R. Someone will abandon.
    end
    t_A=t+exprnd(1/8); % time of next arrival.
    if t_A>1000
        break;
    end
else % upcoming event is departure.
    t=t_D;
    T(i)=t;
    X(i)=X(i-1)-1;
    if M==0 % departure is due to service completion.
        if X(i)==0
            t_D=1000;
            S=1000*ones(N,1);
            % the system is empty, so S should be initialized.
        else
            id=find(S==t_D); %find the one who completed service.
            if X(i)>=N
                %In this case, someone in the queue will get service.
                G=find(R~=1000); %
                d=G(1);
                % Find the first customer waiting in the queue.
                R(d)=1000; % Because he will be served, his
                    % patience time is set as 10.
                S(id)=t+exprnd(1);
                % The service completion time of him.
            else % No one is waiting.
                S(id)=1000;
            end
        end
    end
else % departure is due to abandonment.
    id=find(R==t_D); % find the one who abandoned.
    R(id)=1000;

```

```

        end
        t_D=min(min(S),min(R)); % time of next departure.
        if sum(ismember(S,t_D))==1
            M=0;
        else
            M=1;
        end
        if t_D>1000
            break;
        end
    end
end
plot(T,X,'Color','blue');

Q=max(X-N,0);
for k=1:(i-1)
    inter_T(k)=T(k+1)-T(k);
end
% average of X
ave_X=sum(inter_T.*X(1:(i-1)))/T(i)
% average of Q: average length of queue L_Q
L_Q=sum(inter_T.*Q(1:(i-1)))/T(i)
% average waiting time in queue W_Q
W_Q=sum(inter_T.*Q(1:(i-1)))/N_A
% W=total waiting time/number of arrivals

```

B.3 Simulation of V model

```

%=====
% We take a Markovian queue model as an example to check the feasib-
% ility of this algorithm. We assume that this is an initially empty
% system. We have two classes of customers. Customers of class 2 are
% given priority. Customers of class 1 and 2 arrive according to two

```

```

% independent Poisson process with rate 8 and 1/5 respectively. They%
% are served with exponentially distributed time of mean 1 and 1/2. %
% There are 10 servers in total. The observation time is [0,1000]. %
%=====

close all;
clear all;
t=0;
T(1)=0; % T is the occurring time of events.
X(1)=0; % X is the number of customers in the system.
N=10;
N_A=0;
Q=zeros(2,1);
% Q is the length of queue of customers in two classes.

A(1)=t+exprnd(1/8);
% A(1) is the next arrival time of customers of class 1.
A(2)=t+exprnd(5);
% A(2) is the next arrival time of customers of class 2.
t_A=min(A); % t_A is next arrival time.

S=1000*ones(N,1);
% S is a vector used to save the service completion time.
t_D=min(S);

for i=2:20000
    if t_A<=t_D % upcoming event is arrival.
        t=t_A;
        T(i)=t;
        X(i)=X(i-1)+1;
        N_A=N_A+1;
        if X(i)<=N
            % There is no queue. The newcomer will be served directly.
            Q=zeros(2,1); % The queue is empty.

```



```

F=find(S==1000); % F gives the index of idle servers.
ind=F(1);
% assign the first idle server to the new arrival.
if t_A==A(1) % This newcomer is of class 1.
    S(ind)=t+exprnd(1);
    % service completion time of the new arrival.
    A(1)=t+exprnd(1/8);
    % generate next arrival time of class 1 customer.
else % This newcomer is of class 2.
    S(ind)=t+exprnd(1/2);
    A(2)=t+exprnd(5);
end
else % The new comer will join the queue in this case.
    if t_A==A(1)
        Q(1)=Q(1)+1;
        % Q(1) is the length of queue of class 1 customers.
        A(1)=t+exprnd(1/8);
    else
        Q(2)=Q(2)+1;
        % Q(2) is the length of queue of class 2 customers.
        A(2)=t+exprnd(5);
    end
end
t_D=min(S); % time of next departure
t_A=min(A); % time of next arrival
if t_A>1000
    break;
end
else % upcoming event is departure.
    t=t_D;
    T(i)=t;
    X(i)=X(i-1)-1;
    id=find(S==t_D); % find the one who completed service.
    if X(i)<N % No one is waiting.

```

```

        S(id)=1000; % the corresponding server becomes idle.
        Q=zeros(2,1);
    else % In this case, someone in the queue will get service.
        if Q(2)>0 % If there are customers of class 2 waiting,
            Q(2)=Q(2)-1; % then one of them will get service.
            S(id)=t+exprnd(1/2);
        else % If no one of class 2 is waiting, one of the
            Q(1)=Q(1)-1;%customers in class 1 will get service.
            S(id)=t+exprnd(1);
        end
    end
    end
    t_D=min(S); % time of next departure.
    if t_D>1000
        break;
    end
end
end
plot(T,X,'Color','blue');

q=max(X-N,0); % length of queue
for k=1:(i-1)
    inter_T(k)=T(k+1)-T(k);
end
% average of X
ave_X=sum(inter_T.*X(1:(i-1)))/T(i)
% average length of queue L_q
L_q=sum(inter_T.*q(1:(i-1)))/T(i)
% average waiting time in queue W_q
W_q=sum(inter_T.*q(1:(i-1)))/N_A
% W=total waiting time/number of arrivals

```