January 2019

Level Crossing Simulation of a Queueing Model

Zhanxuan Ding
University of Windsor, ding11y@uwindsor.ca

Level Crossing Simulation of a Queueing Model

by

ZHANXUAN DING

A Major Research Paper
Submitted to the Faculty of Graduate Studies
through the Department of Mathematics and Statistics
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2018

Level Crossing Simulation of a Queueing Model

by

ZHANXUAN DING

APPROVED BY:

A. Hussein

Department of Mathematics and Statistics

M. Hlynka, Co-Advisor

Department of Mathematics and Statistics

P.H.Brill, Co-Advisor

Department of Mathematics and Statistics

December 14, 2018

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify, to the best of my knowledge, that my thesis does not infringe upon anyone's copyright or violate any proprietary rights and that any ideas, techniques, quotations, or any other materials from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted materials that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission of the copyright owner(s) to include such materials in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by committee and the Graduate Studies Office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Simulation of the level crossing method will be used to find approximations of the distribution of the workload for several queueing models. In particular, three different type of queueng models, with different methods of handling workload bound thresholds, will be considered. Simulation applied to workload bound thresholds is new work.

Acknowledgements

First of all, I want to give my sincere gratitude to Dr.Hlynka for his guidance and advices along my way to completing this major paper. His kindness and professionalism always set an example for me to follow and I will take it deep into the heart. I also want to thank Dr.Brill for his incredible guidance on my study, for this paper is based on his previous work on level crossing method. I would also like to thank Dr.Hussein as department reader and it is always great to hear his advices. Finally, I would like thank all of my fellow graduate students for the help I received during my stay in Math and Stat department.

Table of Contents

List of Figures

CHAPTER 1

Introduction

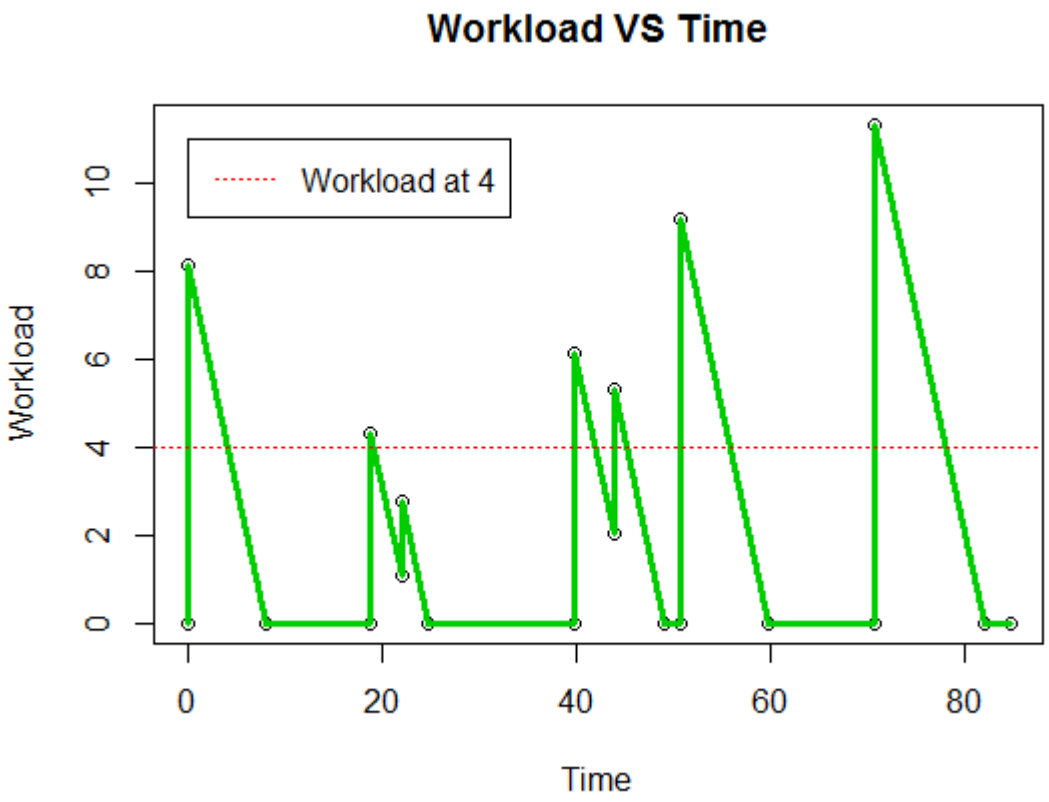
Queueing models have a long history wihtin the probability literatue. Some classic books on the subject include Kleinrock([9], Gross([7]) et al., Gautam([6], Cooper[5], Bolch et al.([2]) , Medhi([10]), Bhat([1]), Jain([8]) et al. Various mathematical methods are used to analyse queues, including differential equations, Laplace Transforms, generating functions, matrix analytic methods, transient queueing methods, fluid methods, and level crossing methods. Level crossing methods were invented by Brill ([3],[4]). They provide a short efficient method of handling certain types of queueing model, and give an intuitive explanation for many results. However, the related integral equations may be hard to set up or may be hard to solve. As a result, simulation of the level crossings can be a useful tool to obtain approximate solutions to various queueing measures.

We begin by considering the simplest nontrivial queueing model, the $M/M/1$ system (independent exponentially distributed interarrivals times and exponentially distributed service times) and show how level crossing methods can be used to find the workload distribution of the system. Assume an arrival rate of λ per unit time, and a service rate of μ per unit time. First we define workload.The workload $W(t)$ at time t is the total time that it will take the server to complete the service of all customers in the system at time t. If a customer arrives at exactly time t, then that customer's service time is included in the workload.We define the virtual wait at time t as the time that it would take to complete service of all customers in the system at time t who arrived before time t. (See Takacs, p. 5) The virtual wait and the workload are identical except at arrival times.

Consider Figure 1.1, which shows the workload for an M/M/1 queue. It shows jumps representing service times and shows diagonal drops with slope -1 indicating

that the workload decreases linearly with time. Pick a level x of the workload, represented by the horizontal line in the diagram.

FIGURE 1.1. Workload vs Time



Brill's level crossing method basically states that the upcrossing rate at level x must equal the downcrossing rate at level x.

Upcrossings will occur if the system is at level 0 and an arrival occurs and the service requirement for the arrival exceeds x. Upcrossings will also occur if the system is at workload level y between 0 and x and an arrival occurs and the service requirement for the arrival is at least $x - y$. Let $f(x)$ be the probability density function for the workload. The complementary cdf of an exponential distribution with rate μ is $P(X > x) = \int_x^\infty \mu e^{-\mu t} dt = e^{-\mu x}$. Thus

$$UpCrossingRate = \pi_0 \lambda e^{-\mu x} + \int_0^x f(y) \lambda e^{-\mu(x-y)} dy.$$

Downcrossings occur if the workoad is at a level just above x and no arrivals occur before the downcrossing. The downcrossing rate is simply the density function.

$$DownCrossingRate = f(x).$$

By equating the two parts, we have

$$f(x) = \pi_0 \lambda e^{-\mu x} + \int_0^x f(y)\lambda e^{-\mu(x-y)} dy. \tag{1}$$

But the long run proportion of time that the system is full is $\rho = \lambda/\mu$ and the long run proportion of time that the system is empty is $\pi_0 = 1 - \lambda/\mu$. Take derivatives (using the fundamental theorem of calculus) to obtain

$$f'(x) = -\mu\pi_0 \lambda e^{-\mu x} + f(x)\lambda + \int_0^x f(y)\lambda(-\mu)e^{-\mu(x-y)} dy. \tag{2}$$

We substitute the expression for the integral from (1) into (2) to obtain

$$f'(x) = -\mu\pi_0 \lambda e^{-\mu x} + f(x)\lambda - \mu(f(x) - \pi_0 \lambda e^{-\mu x}) = (\lambda - \mu)f(x) \tag{3}$$

Thus $f(x) = Ke^{-(\mu-\lambda)x}$ To determine K, we need $\pi_0 + \int_0^\infty Ke^{-(\mu-\lambda)x} dx = 1$. Solving gives $K = (\lambda/\mu)(\mu - \lambda)$

Thus we have the distribution of the workload with an atom at 0 with probability $(1 - \lambda/\mu)$ and an continuous part exponentially distributed with rate $\mu - \lambda$.

Because the task of setting up appropriate equations can be challenging and because the solution techniques needed may be difficult, it is useful to be able to apply the same level crossing modeling with a simulation. This paper will illustrate how this can be done with a set of models related to the one that was just considered.

In this major paper, we discuss some of the features in queueing systems with a single server.

In chapter 2, a number of queueing measures are defined and discussed. The key quantities are interarrival time and service time. These two quantities give us information about arrival times, number of customers in the system, customer

waiting time, and workload. All quantities to be discussed are based on interarrival times and service times.

After generating vectors of interarrival times and service times by computer simulation, we are able to calculate quantities of interest: idle times, remaining [workload] time. Then we study relationships among these quantities. Further, we create algorithms to obtain workload and frequency in a customer-intense context, draw illustrative diagrams for workload versus time, as well as frequency and density graphs of the workload. The M/M/1 setting is used here.

In the third chapter, we allow the interarrival and service time to follow different distributions (other than exponential) so that queueing systems can be generalized as G/G/1. We simulate interarrival and service data for various distributions, then plot diagrams as before and study the changes of these diagrams compared to the M/M/1 case.

In the fourth chapter, we concentrate on cases where workload is bounded. Three types of methods for handling such cases are proposed: (a) reject access to service; (a) give partial access to service; (c) give full access to service but restrict further arrivals. Each method requires its own algorithm. The bound will help reduce situations of overload, though the three different methods have different impacts on workload.

In chapter 5, we give a method to approximate the distribution of workload. We create an algorithm to find the counts at different levels of x from computer implementation, then transfer the counts into probabilities. In addition, applying resampling techniques, we refine our pdf to a better approximation.

Chapter 6 gives conclusions.

CHAPTER 2

Queueing Measures

2.1. Inter-arrival Time and Service Time

In a queueing model, inter-arrival time refers to the amount of time between two consecutive customers coming into the system. Service time denotes that the amount of time the server has to spend on a particular customer.

2.2. Workload

Workload at time t is a term used to describe the amount of time a server has to spend on customers in a queueing system at time t. A server's workload will change over time; the workload will depend on the number of customers in the system and their respective service needed.

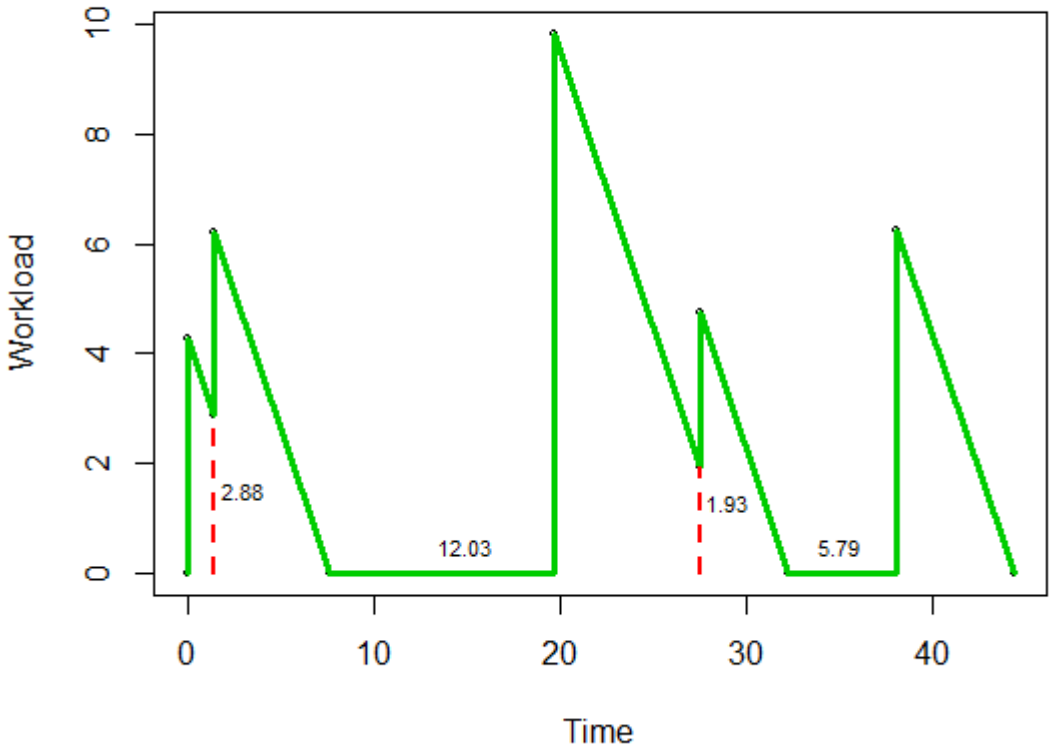
At a point in time when there is no customer in the system, the workload is equal to 0. Once a customer enters the system, the workload will jump by a quantity equal to the service time of this customer. At that point in time, the workload will have a linear decline with a slope of -1 until the next customer arrives or the workload hits zero. For example, at intial time 0, assume a customer enters the system. The workload will immediately jump right to an amount equal to the service time of this customer and then decrease linearly. If the next customer arrives before the workload hits zero, then it jumps. If the workload hits zero before the next customer arrives, then it will stay at 0 until the next customer enters the system.

2.3. Idle Time and Remaining Time

In a queueing system, we assign to each customer C_i an idle time IT_i . For example, if our queueing system has 10 customers, then we will have 10 idle times assigned to each customer. The idle time IT_i is the length of time that the server

has been empty at the time that customer C_i enters the system. In Figure 2.1, we see that the first customer arrives at time 0, idle time is assigned 0, so $IT_1 = 0$. Then for the second customer, because there is a period of time 6.89 that the server is empty (workload is zero) before it enters, the idle time for the second customer is $IT_2 = 6.89$. Moving on, we assign 0 to the third customer because there doesn't exist a length of time that the server is empty, $IT_3 = 0$. Thus, according to this rule, we can assign each customer an idle time.

FIGURE 2.1. Workload. The server is idle when workload is zero.



In a single server system, as customers arrive, the server will have different statuses, namely busy and idle. These two status are given to the server when a new customer arrives. The server is considered to be busy when a new customer arrives while the server still is working on another customer. On the other hand, if the server has already completed its work on all customers before a new arrival, the server will be considered to be idle just before this customer arrives.

We assign the idle time for a new customer coming into the system to 0, $IT = 0$, if the server's status is busy by the time this customer arrives. So if the server has other previous customers left to work on, then the idle time for a new customer will be 0 since after completing its service on those previous customers, the server has to immediately serve this new customer. On the other hand, the idle time for a new customer will be non-zero if the server's status is idle when this customer arrives, i.e. $IT \neq 0$.

Suppose we have 1000 customers entering a queueing system. each customer will have its required service time $S_i, i = 1, 2, ...1000$. We assume that the first customer arrives at time 0, so there will be 999 inter-arrival times between customers, we denote them as $I_i, i = 1, 2, ...1000$, with $I_1 = 0$. If we know the total number of customers and service times as well as the inter-arrival times, we will be able to calculate idle time for each customer.

In order to get a vector of idle time $IT = (IT_1, IT_2, \ldots, IT_{1000})$, we define another quantity RT, remaining time, meaning that the amount of time a server has yet to work on previous customers at the time a new customer comes. The difference between workload and remaining time is 0 excerpt that at arrival times. At arrival times, the workload includes the time of the new arrival while the remaining time does not. The remaining time is useful not only in obtaining idle time but also in calculating workload later on. As the definition shows, RT_1 is equal to 0 since the server does not have any work to do when the first customer comes. On the other hand, if the server still has work to do with preceding customers when a new customer arrives, then we consider remaining time to be equal to the amount of time it has left.

2.4. Relations Among Workload, Idle Time and Remaining Time

At time 0, the first customer enters the system. Hence idle time for the first customer is 0, $IT_1 = 0$, and workload at this point of time jump from 0 to S_1, and the server will complete service on the first customer at time S_1. If the second customer enters the system before time S_1, then the idle time for the second

customer will be 0 because the server still has work to do with the first customer at time when the second customer comes so RT_2 will be great than 0. If the second customer enters the system after time S_1, then the server will finish its service on the first customer and wait until the second customer shows up. In this case, the idle time for the second customer will be (I_1 - S_1) and $RT_2 = 0$. Then we follow the same rule to obtain $IT_3, IT_4, \ldots, IT_{1000}$ as well as $RT_3, RT_4, \ldots, RT_{1000}$.

Therefore, we can obtain idle time and remaining time for each customer as follows:

Step 1

$$IT_1 = 0$$

$$RT_1 = 0$$

Step 2

$$IT_{n+1} = max(I_n - S_n - RT_n, 0)$$

$$RT_{n+1} = max(S_n + RT_n - I_n, 0)$$

n = 1, 2, ..., 999

Once we figure out the idle time for each customer from a given set of inter-arrival and service time, we can then move on to obtain workload at those arrival times of new customers.

For example, when we have a set of inter-arrival and service time as follows:

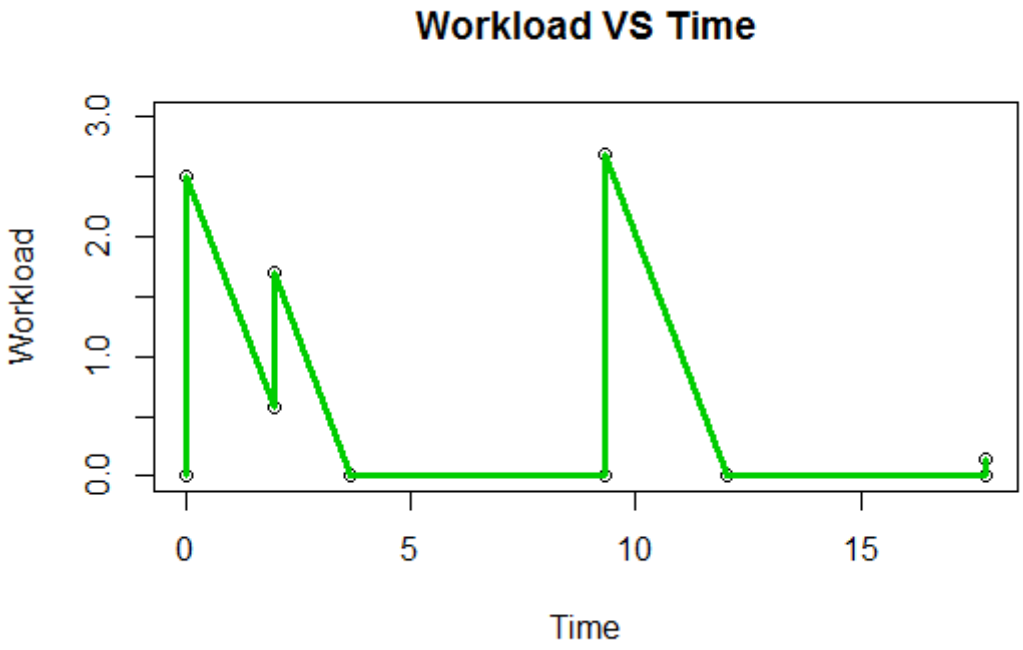
$$I = (1.94,\ 7.39,\ 8.47,\ 0.18) \quad (4)$$

$$S = (2.51,\ 1.13,\ 2.69,\ 0.14,\ 0.23) \quad (5)$$

Our time versus workload relationship can be shown as follows:

Relationship between workload and time is shown in Figure 2.2 with inter-arrival and service time as given in the example.

FIGURE 2.2. Workload versus time



Notice that when we obtain IT_i and RT_i, we find that the workload is always associated with RT_i, specifically,

$$Workload_i = S_i + RT_i$$

When the i^{th} customer arrives, the workload is the sum of service time S_i and remaining time RT_i of the i^{th} customer.

Meanwhile, one can observe the plot is comprised of "mountain" shape lines and they all have the same slope after reaching their peaks. We can prove that for any given set of inter-arrival and service time, the slope of mountains will always be -1. The reason is that for the i^{th} customer, its workload peak is $(S_i + RT_i)$ and if there is no more customer coming into the system, the service is destined to be completed at time $(Ar_i + S_i + RT_i)$. Ar_i denotes the point of time when the i^{th} customer arrives. Therefore, to obtain the slope of each "mountain", one will have to know coordinates of two points in a workload-time coordinate graph: peak and destined complete time. From above analysis, we can obtain the coordinates of

these two points:

$$(Ar_i,\ S_i + RT_i)$$

$$(Ar_i + S_i + RT_i,\ 0)$$

Therefore,

$$\begin{aligned} Slope &= \frac{(S_i + RT_i) - 0}{Ar_i - (Ar_i + S_i + RT_i)} \\ &= -1 \end{aligned}$$

Since the value is fixed, it means the slope is fixed no matter the value of i.

After determining workloads at arrival times, we need to know workload during inter-arrival times between two customers. First we notice that the workloads are always on a linear decline towards 0 during inter-arrival periods and have the same slope of -1, which makes it easier for us to measure the relation between workload and time. However, one has to know that there are two scenarios need to be considered. One is that workload would drop to 0 before the next customer arrives and would remain unchanged until the next customer shows up. The other one is that workload is on its declining way to 0 but does not reach 0 when the very next customer shows up. In this scenario, workload will immediately increase by the amount of service time of the next customer.

In an attempt to create an intuitive interpretation in terms of the relationship between workload and time, one can draw a coordinate graph as shown above. It is easy to plot it when we have only 5 customers coming into the system, but in the case of 1000 or even more, we need to figure out a way to use computer programming method to present.

First, one can observe that to plot such graph, we don't need the coordinate of every point on X-axis, since it is a time series and X-axis time is continuous. What we need here is those points when either workload hits zero or a new customer arrives. Once we figure out the coordinate of those points, we are able to plot them on a graph.

Finding these critical points when either workload hits zero or a new customer arrives is relatively time-saving. Since we already obtain idle time and remaining time, we may use them to find these coordinates, here is the algorithm:

(1) set T_1, T_2 equal to zero, $T = (T_1,\ T_2)$; set Wl_1 equal to zero and Wl_2 equal to S_1, $Wl = (0,\ S_1)$.

(2) if IT_i is nonzero, let

$$\begin{aligned} T &= (T,\ Ar_i - IT_i,\ Ar_i, Ar_i) \\ Wl &= (Wl,\ 0,\ 0,\ S_i) \end{aligned}$$

(3) if IT_i is zero, let

$$\begin{aligned} T &= (T,\ Ar_i,\ Ar_i) \\ Wl &= (Wl,\ RT_i,\ S_i + RT_i) \end{aligned}$$

(4) $i = 2, 3, ..., n$

With this algorithm, we are able to find the relationship between server's workload and time. Each element in the time vector denotes a point of time when either a new customer arrives or workload hits 0. As we thread these point on the two dimensional coordinate, we can get a clear and dynamic relation between workload and time.

Two vectors of workload and time with 100 elements are plotted at times when workload reaches peaks or hits zero (first), and fitted line on them (second). Workload and time are obtain through simulated dataset of inter-arrival and service time from exponential distribution.

From the second graph, we can have a clear view of what workload will look like over time if given a set of inter-arrival and service time. However, sometimes we may want to know workload of a particular point of time, not those times when a customer shows up or workload drops to 0. The two vectors of workload and time cannot give us necessary information about workload at these points of time.

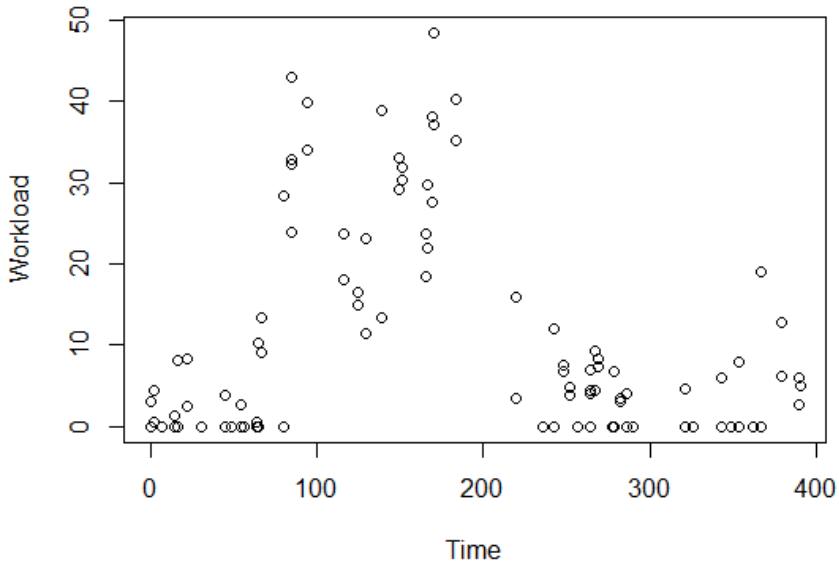


FIGURE 2.3. Workload endpoints

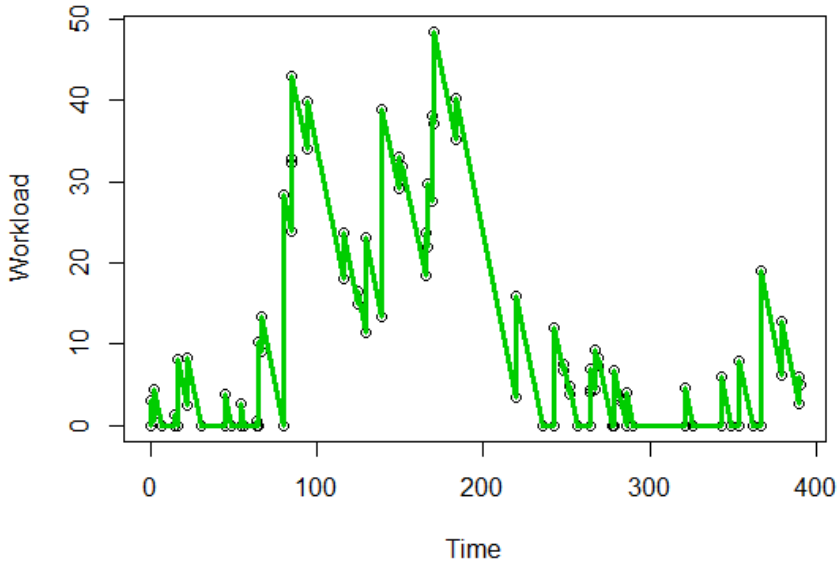


FIGURE 2.4. Workload

According to results above, it is obvious that there are the values of remaining time and workload are different at times when customers arrive. We call the

smaller of the two lower bound and the larger one upper bound. The following is the algorithm to get workload on a specific point of time:

(1) Obtain lower bound at arrivals

  (a) let Lb_1 equal to S_1

  (b) if IT_i is non-zero, let $Lb_i = 0$

  (c) if IT_i is zero, let $Lb_i = RT_i$

  (d) $i = 2, 3, ..., n$

(2) Obtain function of workload when given a value of *time*

  (a) if $Time <= Ar_2$, then

$$Workload = -Time + Lb_1$$

  (b) if $Time <= (Ar_{i+1} - IT_{i+1})$, then

$$Workload = Lb_i + Ar_i + S_i - Time$$

  (c) if $Time <= Ar_{i+1}$, then $Workload = 0$

  (d) $i = 2, 3, ..., n$

Once we apply the function above, we can write a program so that given a value of time, we obtain the corresponding workload.

2.5. Frequency/Count

From the previous discussion we have found the relationship between workload and time once given a set of inter-arrival and service time. Now we would like to know the number of times or the count or frequency of a server being at a certain level of workload throughout the whole time series generated by the customer interarrival and service times. For example, we would like to know how many times a server has had a workload of level 20 units, measured from time 0 to final completion of service of all customers.

Knowing the relationship between frequency and workload is very useful because we would like to know the work intensity of a server. If workload is high

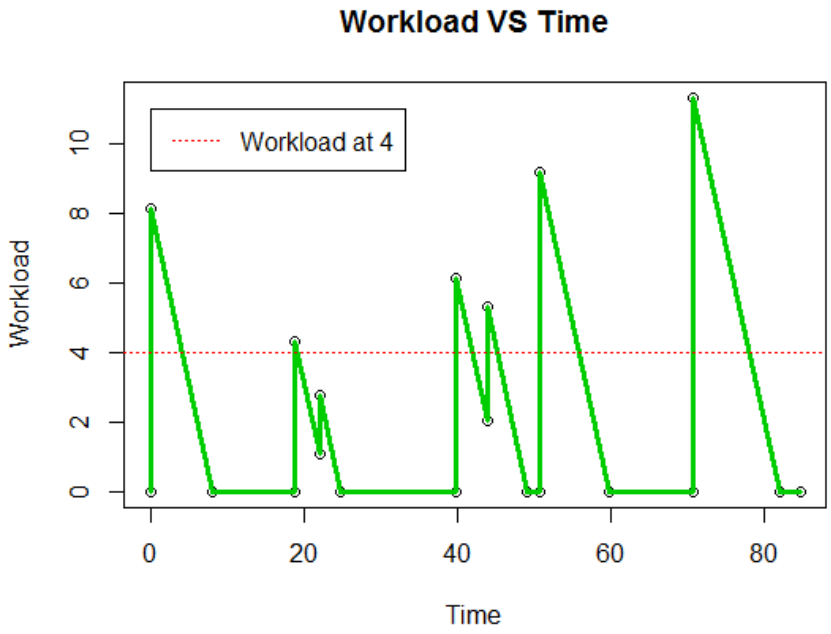
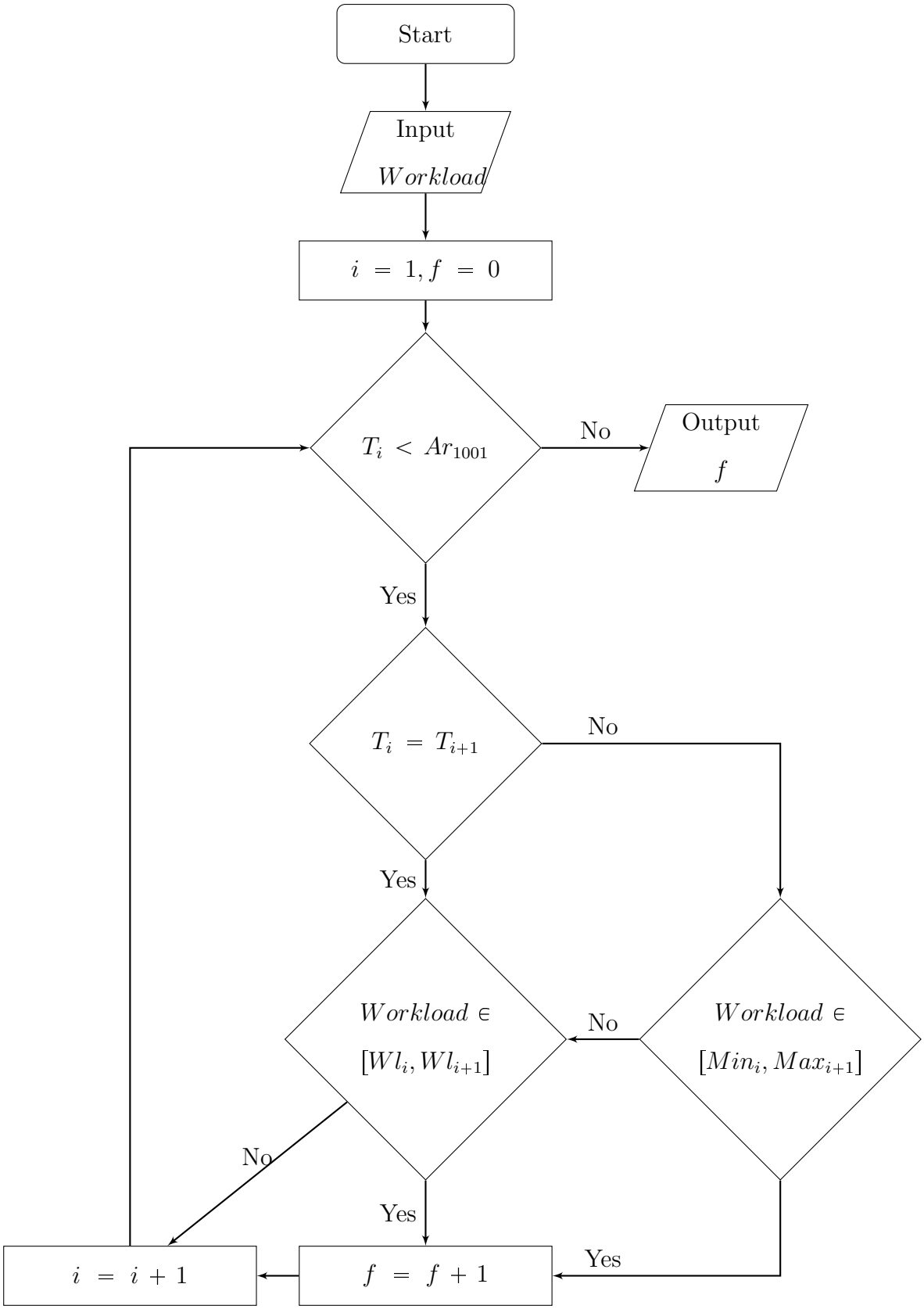


FIGURE 2.5. The frequency of a server having a workload of 4 is 12.

and its corresponding frequency is also relatively high, that would mean that the server is consistently under high workload pressure, and we may want to relieve its workload by some method as discussed later. On the other hand, if frequency is low when workload is high or frequency is high when workload is low, it appears the server may have a leisurely working environment.

It is easy to count when we have a small set of customers in our system, while we may encounter difficulty doing that for hundreds of thousands of customers in a queueing system.

However, we are able to obtain frequency given a value of workload based on the following flowchart:



Note: $Min_i = Min(Wl_i, Wl_{i+1})$ and $Max_{i+1} = Max(Wl_i, Wl_{i+1})$

CHAPTER 3

Different Queueing Models

In the previous chapter, we discussed wait, remaining time and workload as well as frequency. We used algorithms to obtain these quantities. However, these quantities are derived from customers' original dataset of inter-arrival and service time. In other word, we have a prior information about inter-arrival and service time among customers. However, in reality, we may not be able to have access to this information beforehand and it is also extremely time-consuming to collect data when we have a lot of customers entereing a queueing system.

Instead of collecting data, we would like to study different distributions of customers' inter-arrival and service time. In this way, it is more relevant to simulate the behaviour of customers in a particular queueing system and easier for us to try to find a pattern among customers. Hence, if inter-arrival and service time follow particular respective distributions, we are able to simulate a dataset based on their distributions to study those quantities discussed in the previous chapter.

3.1. Birth and Death Process

As discussed above, inter-arrival and service times with exponential distributions form a birth-death process. The birth-death process is a special case of continuous-time Markov process where the state transitions are of only two types: "births", which increase the state variable by one and "deaths", which decrease the state by one. The process goes from state n to $n+1$. When a death occurs, the process goes from state n to state $n-1$. The process is specified by birth rates μ_i and death rate λ_i, $i = 1, 2, ..., n$.

We consider birth-death process in a perspective of interarrival and service time. We have μ as the arrival rate of customers per unit time. The reciprocal $\frac{1}{\mu}$ gives the average amount of time for i^{th} customer to come after $(i-1)^{th}$ customer

arrived. Therefore, $\frac{1}{\mu}$ can also be viewed as the average inter-arrival time between i^{th} and $(i-1)^{th}$ customer. We can also apply λ_i to understand the relationship between death rate and service time.

M/M/c model is used in queueing theory. It is a birth-death process used to describe customers in an infinite queue. M refers to exponentially distributed interarrivals and service times. c denotes the number of servers in the queueing system. Therefore, M/M/1 is a queueing system where customers' inter-arrival and service time follow exponential distributions with $\mu_i = \mu, \lambda_i = \lambda$, with one server is in the system.

Once we know the distribution of arrivals and departures and the number of servers in the system, we are able to randomly generate a dataset of inter-arrival and service time to simulate the real situation of customers' arrivals and departures. For example, if we know a queueing model is $G_1/G_2/1$, where G_1 refers to uniform distribution $unif(\alpha, \beta)$ and G_2 refers to an exponential distribution with parameter μ, then we can generate a dataset of interarrivals from a uniform distribution and a dataset of service from an exponential distribution.

Given the set of interarrival and service times, we are able to the quantities discussed the previous chapter such as idle time and workload as well as frequency.

3.2. Frequency Displayed in Different Queueing Models

We would like to study the relationship between frequency and workload in different queueing models with a single server system.

3.2.1. M/M/1. In this single server queueing model, we discuss a scenario where the arrivals and departures both follow a exponential distribution with parameter λ and μ, $\lambda < \mu$. We generate a dataset for inter-arrivals and services from their respective distributions to simulate the real situation of 1000 customers entering the system. So we have our simulated dataset $I = (I_1, I_2, ..., I_{999})$ and $S = (S_1, S_2, ..., S_{1000})$. According to our previous algorithm, we can obtain a relationship between workload and time, frequency and workload.

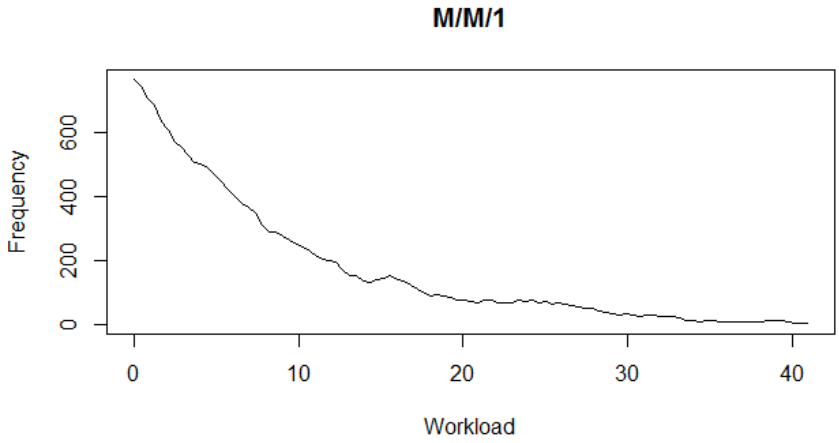


FIGURE 3.1. M/M/1 Workload

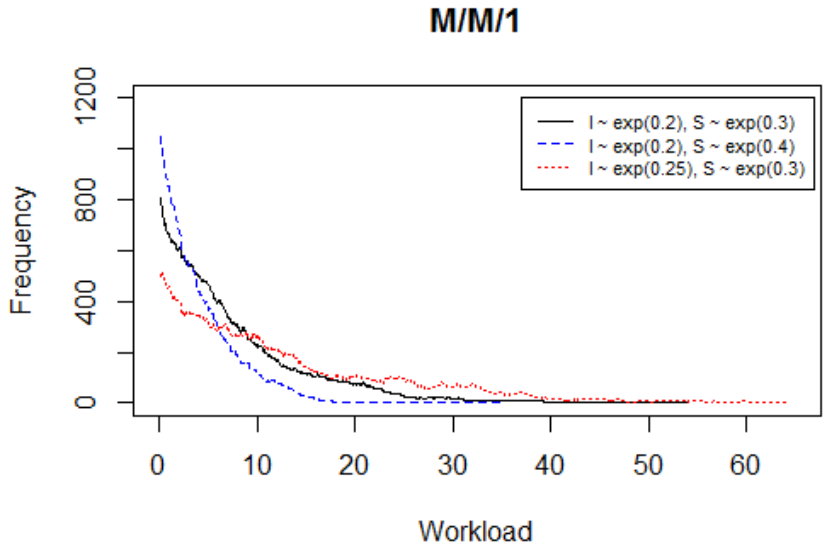


FIGURE 3.2. M/M/1 Workload. Different paramters.

Figure 3.1 shows a plot of frequency with respect to workload, data of inter-arrivals generated from $I \sim exp(0.2)$ and services from $S \sim exp(0.3)$.

As the Figure 3.1 shows, frequency tends to decrease monotonically as workload increases. When workload reaches its maximum around 40, frequency hits 0. Also, Frequency drops dramatically as workload augments from 0 to 10 and then starts to slow down. When workload increases from 30 to 40, changes of frequency become minimum.

Curves of frequency with respect to workload, derived from three datasets of inter-arrival and service time are shown in Figure 3.2, generated from three M/M/1 models with different set of parameters.

As far as Figure 3.2 is concerned, it is noticeable that when we increase μ and keep λ intact, the curve (blue) will have a higher upside but shorter tail. On the other hand, if we increase λ but keep μ unchanged, the curve (red) will have a lower upside but longer tail. This makes sense since if a server speeds up its service on customers, it would encounter fewer situations of high workload as blue curve suggests. If customers come into system less frequently, the server would face fewer situations of high workload as the red curve shows.

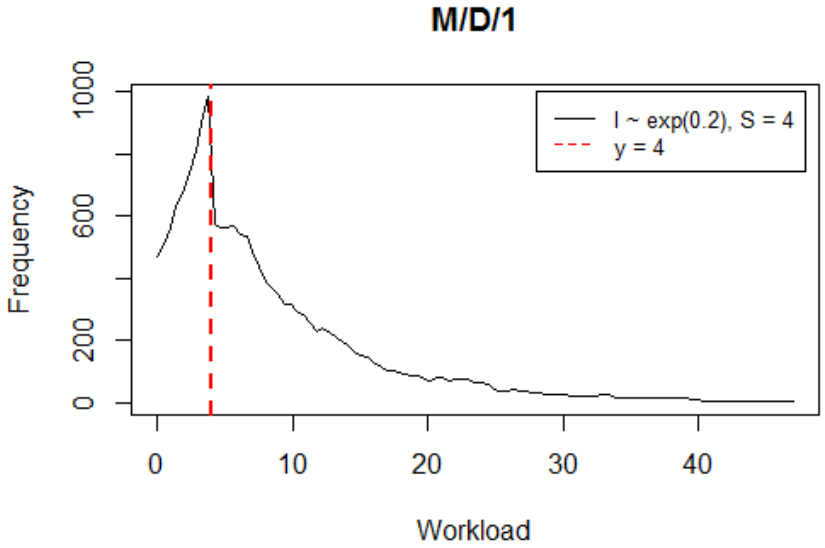


FIGURE 3.3. Frequency and workload in a M/D/1 model

3.2.2. M/D/1. In the previous subsection, we know that frequency declines monotonically as workload increases in the M/M/1 model. Now we want to exam the curve under an M/D/1 model. Here, D refers to a deterministic service time, meaning the server will always complete its service in a fixed amount of time on every customer.

As the figure suggests, when the service time becomes deterministic, it is noticeable that frequency reaches its peak right before workload increases to the value equivalent to the deterministic value of service time, which is 4. Once workload breaks through 4, frequency plummets, then slows down and continue to decline. The main reason why frequency at workload of 4 is so high is that when a server turns to an idle status, its workload will jump to the deterministic value once a new customer arrives. Therefore, the more idle status a server has, the more likely its frequency at workload of 4 is so workload become higher.

CHAPTER 4

Queueing Models With Bound on Workload

In the previous chapter, we discussed the relationship between workload and time as well as frequency, without conditions on workload. However, in this chapter, we are going to discuss scenarios where a limit is imposed on workload and we consider the respective shape of frequency curve as well as the pdf of workload based on the frequency curve. In addition, we use resampling methods to smooth our frequency curve and the resulting pdf such that we will have a more accurate description about the workload.

4.1. Partial Access to Service

For a particular queueing system, sometimes we may want to make sure that the server will not encounter high volume of workload. So it is reasonable to set a limit on workload, say B, we want to make sure the server's workload can reach, at most, but will not exceed B. If a workload would, at certain point of time, exceed the bound value B, then we will only allow partial access to service such that the workload will only increase to B rather than exceed it.

In this case, workload has to be reduced to B once it exceeds that bound, so we may need to adjust some of service times in which workload would have exceeded the bound had we not carried out this step. For example, assuming that by the time t i^{th} customer arrives, the server's workload is Wl_t ($Wl_t < B$) and the service time for this customer is S_i. If

$$Wl_t + S_i > B$$

we will reset S_i equivalent to $B - Wl_t$ so that every customer will get full service if workload is not over the bound after adding the amount of service time or get partial access to service to reach a bound value of workload if workload would

have been over the bound. Hence, by doing this procedure, we will change some of elements in the vector $S = (S_1, S_2, ..., S_n)$, which again lead to another change in the way we obtain idle and remaining time. Therefore, the method to get wait and remaining time in previous chapter is no longer valid in this case.

Algorithm 3.1 below gives us a way to obtain idle time, remaining time and workload:

(1) Input inter-arrival and service time.

(2) Set initial values for critical points of time when a new customer arrives or workload hits zero, and its corresponding workload as well as remaining time for each customer:

$$T_1 = Wl_1 = RT_1 = 0$$

(3) Set a bound value for workload B.

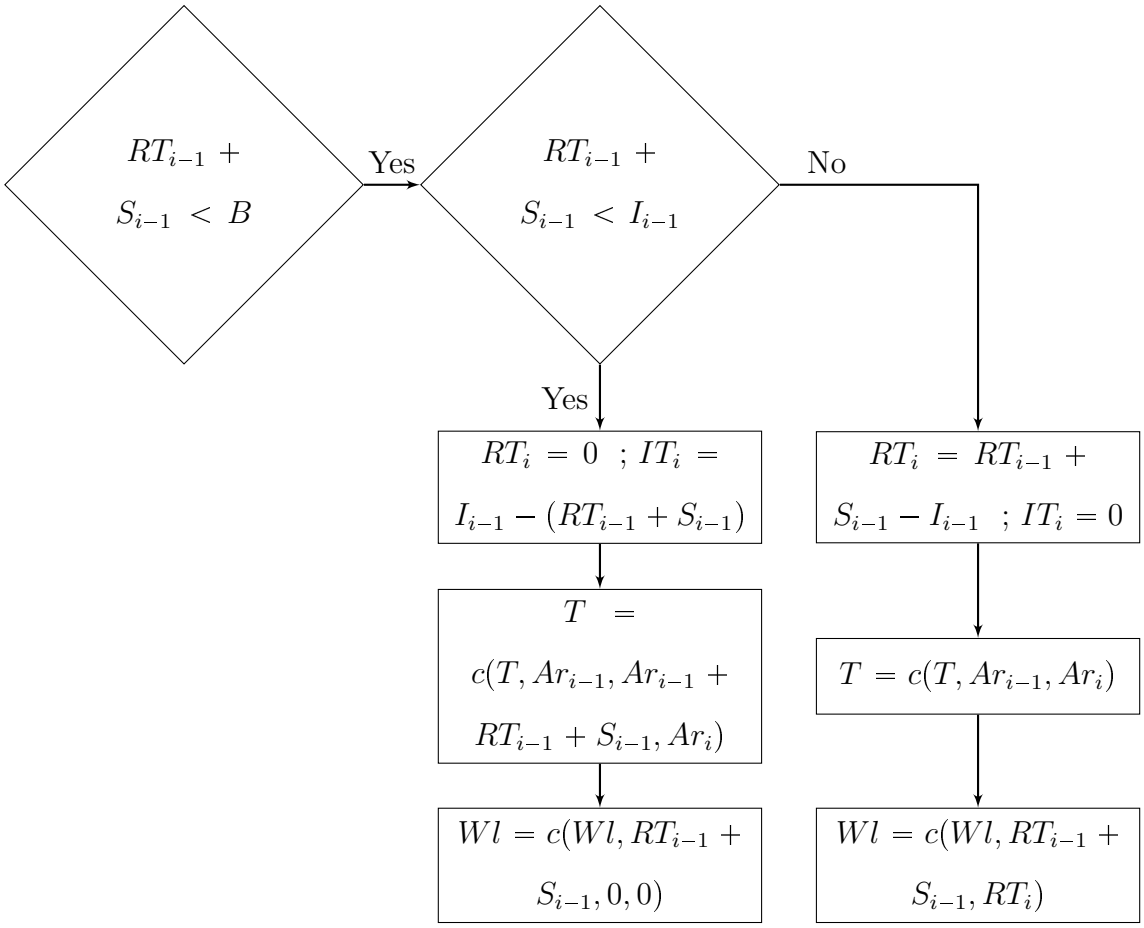
(4) Use loops to obtain critical points T and Wl along with IT and RT.

(a) Let N be the total number of customers and i equal to 2

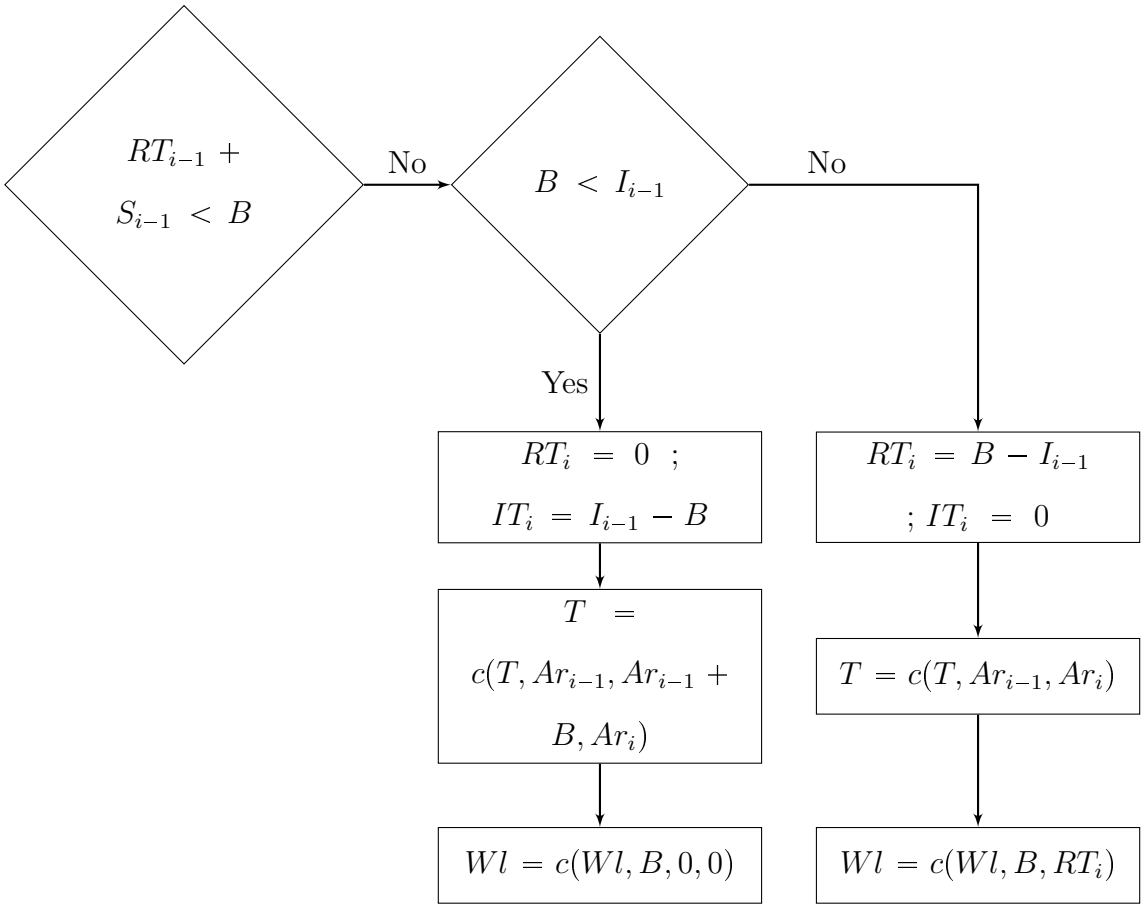
(b) Test if $i < (N + 1)$ holds, if no, then

$$\begin{aligned} T &= c(T,\ Ar_N,\ Ar_N + min(B,\ RT_N + S_N)) \\ Wl &= c(Wl,\ min(B, RT_N + S_N),\ 0) \end{aligned}$$

(c) If the formula in (b) holds and $RT_{i-1} + S_{i-1} < B$ holds too, then we will follow the below procedures:



(d) If the formula in (b) holds but formula $RT_{i-1} + S_{i-1} < B$ in (c) does not hold, then we will follow this procedure below:



(e) Let i increase by 1, $i = i + 1$, then go back to (b)

(f) Jump out of the loop when $i = (N + 1)$.

(5) Output critical points of time and its workload, T and Wl, along with wait and remaining time, IT and RT.

(6) Stop.

The above algorithm will make sure a server's workload never exceed the bound value that we set. If at some points of time workload tends to exceeds that bound, the algorithm will set the workload of these points of time as the bound value, then adjust the corresponding service time to the bound value as well.

By using the above algorithm, we are able to obtain the critical points of time and its corresponding workload, then we apply the algorithm of 1.5 to get a relationship between frequency and workload given a bound on workload.

Figure 4.1 shows workload with respect to time in a M/M/1 system containing 10 customers before and after we set a bound on workload.

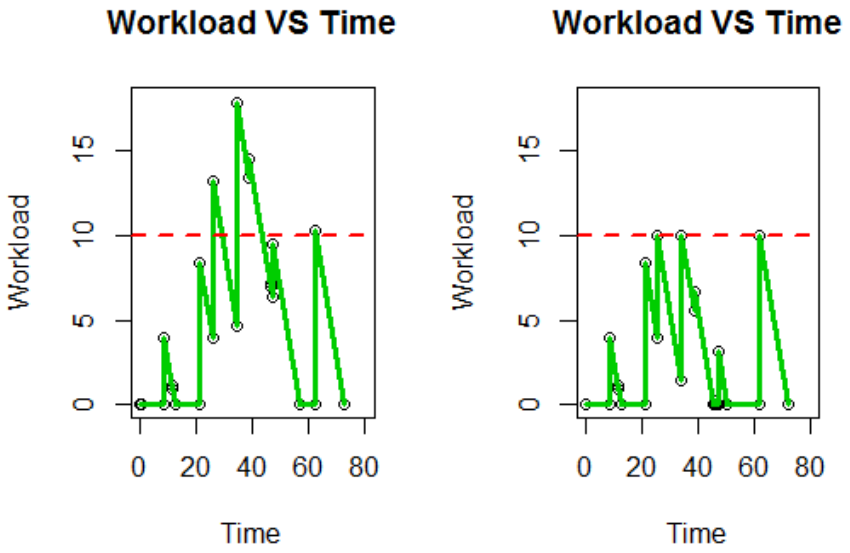


FIGURE 4.1. Workload with Bound.

The figure shows that once we add a bound on workload the relationship between workload and time will change dramatically. Once workload first exceeds the bound value, which is 10 above, the shape or dynamic of workload will differ since then. If we compare the server's total idle time between these two scenarios, the one with a bound on workload tend to have more idle time, 30.26 units of idle time for the right plot comparing with 22.41 units for the left, which further validate that adding a bound on workload will ease a server's work and increase more idle time for a server.

On the other hand, server of a queueing system with limits on workload tend to complete its service on all customers no slower than the one without limits on workload. Since in workload-limited systems, a server will complete its service at time $(Ar_N + min(B,\ RT_N + S_N))$ while service is not completed until at time $(Ar_N + RT_N + S_N)$ in systems without limits on workload.

Figure 4.2 shows the relationship between frequency and workload as we change the value of bound on workload in a M/M/1 system containing 1000 customers in which $I \sim exp(0.1)$ and $S \sim exp(0.15)$.

In cases where workload is unlimited, frequency will drop all the way as workload increases, but slowly decline to zero as workload approaches its maximum.

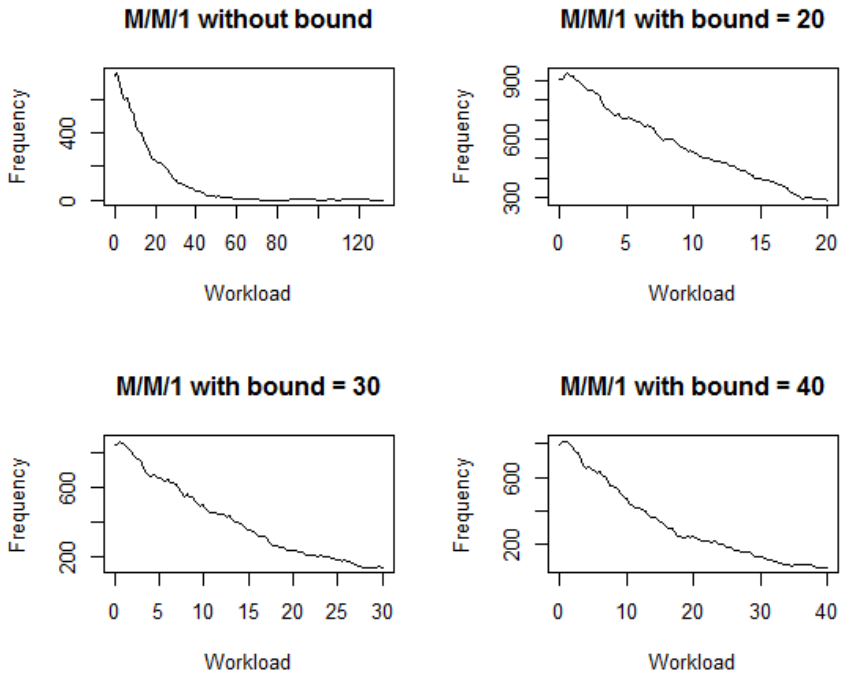


FIGURE 4.2. Frequency and workload as bound changes

However, if a bound is imposed on workload, frequency will plummet as workload approaches the bound value but will not usually hit zero.

In addition, we observe that once we impose a bound on workload, the frequency curve is always shorter but higher than the one without bound on workload. As bound value increases, the frequency curve will become longer and lower, but always above the curve without bound.

4.2. Reject Service

Next we present a second strategy to deal with limitation on workload. Unlike the one we mentioned in the previous section, now we will apply a different method: if the workload exceeds the bound value that we set as a new customer arrives, we decide that this customer will not be served at all, and the workload will continue to drop until next customer comes. No matter when customers show up, the server will reject service to these customers whose presence increases the workload over the bound value. If a customer is to be rejected, then the workload will drop at the same pace as it does before this customer shows up.

To solve this problem, we come up with a similar algorithm as the one in section 3.1:

(1) Follow all procedures of Algorithm 3.1 except (b) and (d) of Step 4

(2) Set a new variable $reject$ equal to zero, $reject = 0$

(3) We change the (b) of Step 4 as follows: Test if $i < (N + 1)$ holds, if no, then

(a) if $RT_N + S_N > B$, then

$$\begin{aligned} reject &= reject + 1 \\ T &= c(T, Ar_N + RT_N) \\ Wl &= c(Wl, 0) \end{aligned}$$

(b) if $RT_N + S_N <= B$, then

$$\begin{aligned} T &= c(T, Ar_N, Ar_N + RT_N + S_N) \\ Wl &= c(Wl, RT_N + S_N, 0) \end{aligned}$$

(4) We change the (d) of Step 4 as well: If the formula in (b) holds but formula $RT_{i-1} + S_{i-1} < B$ in (c) does not hold, then we will follow this procedure below:

(a) if $RT_{i-1} < I_{i-1}$, then

$$\begin{aligned} reject &= reject + 1 \\ RT_i &= 0 \\ IT_i &= I_{i-1} - RT_{i-1} \\ T &= c(T, Ar_{i-1} + RT_{i-1}, Ar_i) \\ Wl &= c(Wl, 0, 0) \end{aligned}$$

(b) if $RT_{i-1} >= I_{i-1}$, then

$$reject = reject + 1$$

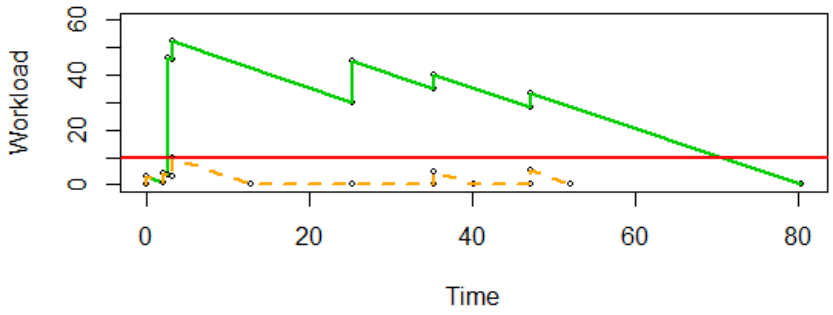


FIGURE 4.3. Workload Frequency with Rejected Customers

$$
\begin{aligned}
RT_i &= RT_{i-1} - I_{i-1} \\
IT_i &= 0 \\
T &= c(T, Ar_i) \\
Wl &= c(Wl, RT_i)
\end{aligned}
$$

After implementing this algorithm in R, we will be able to plot a diagram of workload with respect to time. The variable *reject* is used to record the number of customers being rejected from service.

Suppose we have 7 customers coming into the M/M/1 system with $\mu = 0.15$ and $\lambda = 0.1$ in which the inter-arrival time is I = (1.934, 0.583, 0.633, 22.114, 10.035, 11.734) and its corresponding service time is S = (2.874, 3.437, 42.478, 6.545, 15.200, 4.761, 5.020).

Figure 4.3 shows 7 customers arriving. Green lines: workload without bound. Red line: Bound value at 10. Yellow dashed lines: workload with bound, reject service when workload exceed bound.

If we are to reject customer service when workload exceeds the bound, as shown in Figure 4.3, workload will tend to be lower and shorter because customers with high service time will be prevented from being served. In this case, with this method applied, we reject the third customer and other customers are served. The

server's workload is kept low and it will also finish the job earlier than that without a bound.

The another effect of this method is that server will gain more idle time over the whole period. In Figure 4.3, the server would have no idle time until all 7 customers have completed their service at 80.31 while with this condition, server will have 29.42 units of idle time out of total 52.05 units, meaning $29.42/52.05 = 56.5\%$ of time the server is in a idle state and only one is rejected from service.

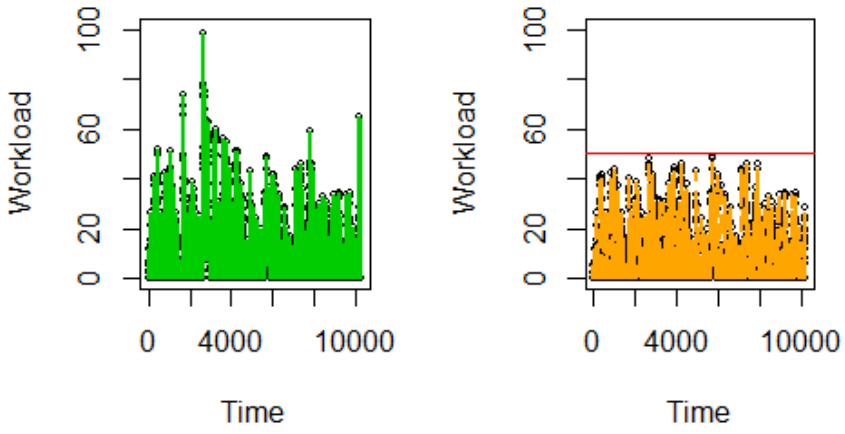


FIGURE 4.4. Frequency with high arrival rate

Now we want to exam if this method have an effect on server's idle time when there are a large number of customers arriving in the system. Suppose we have 1000 customers comes into the M/M/1 system with $\mu = 0.15$ and $\lambda = 0.1$, and we set workload bound at 50. We will get a workload plot as below:

Figure 4.4 (Left) shows Workload without a bound, while Figure 4.4(Right) shows Workload with a bound at 50.

Compared to the case without a bound, the right diagram of Figure 4.4 has a significantly low workload throughout the whole period. In our simulation, we have 14 customers rejected from service and the server is idle 39% of the time comparing with 36% without a bound. We may conclude that by rejecting certain service-heavy customers from receiving their service, we are able to chop down the server's workload effectively and give it more idle time.

Figure 4.5 shows frequency curves obtained by applying method of rejecting access to service in cases where different bounds on workload are imposed.

As Figure 4.5 shows, if we set a bound at workload of 40 and consider workload near 40 is overload, then the green curve indicates that the server will encounter fewer overload situations. Thus, it is illustrative that when we apply method of

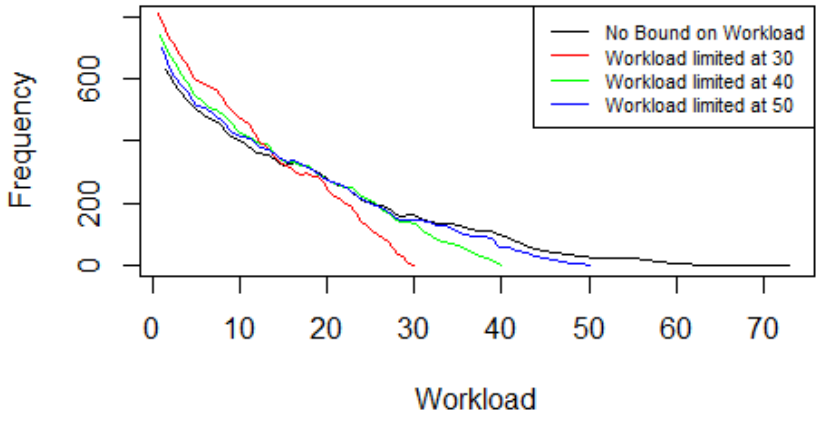


FIGURE 4.5. Different Bounds

rejecting service given a certain workload bound, it helps the server to reduce situations of heavy workload.

4.3. Full Access to Service

Finally we will apply another a third method to deal with limit on workload. In this scenario, we allow service for customers whose presence increases the workload from below the bound value to exceed it, and any customers come next at a time before the workload drop below the bound value will be rejected from service. That is to say, once the workload exceeds the bound value, the next customer to be served has to come after the workload is below the bound.

As Figure 4.6 shows, the arrival of the second customer increases workload over the bound, but the system still give this customer full service, then the workload decreases below the bound until the third customer arrives. What we should notice is that when workload reaches its peak at 26.6 and starts to decline, there is a customer arriving while workload is still above the bound. By the method discussed in this section, we reject this customer from gaining its service and workload continues dropping as shown in Figure 4.6.

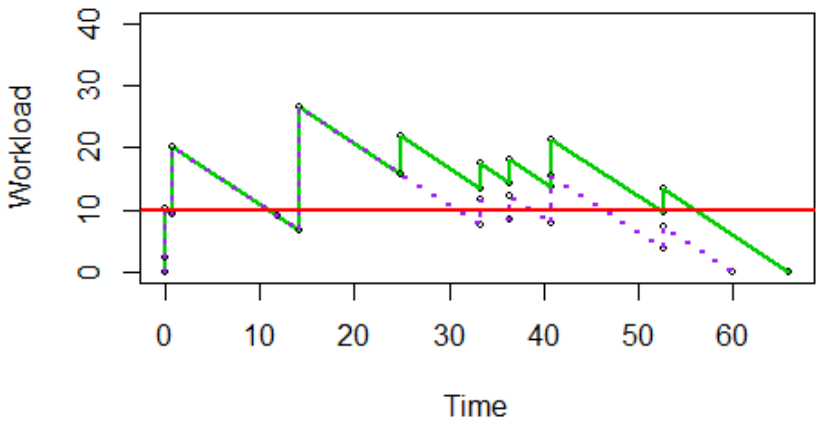


FIGURE 4.6. Frequency workload in Full Access model

In Figure 4.6, Green lines show workload without a bound, Red line shows bound value at 10. Purple dashed lines show workload where customers are accepted to full access to service only when workload is below the bound.

Now we implement this method in R and the following is the algorithm:

(1) Follow Step 1, 2 and 3 of Algorithm 3.1

(2) Set a new variable $reject = 0$

(3) Follow (a) and (c) of Step 4 in Algorithm 3.1

(4) Revise (b) of Step 4 in Algorithm 3.1: Test if $i < (N + 1)$, if no, then

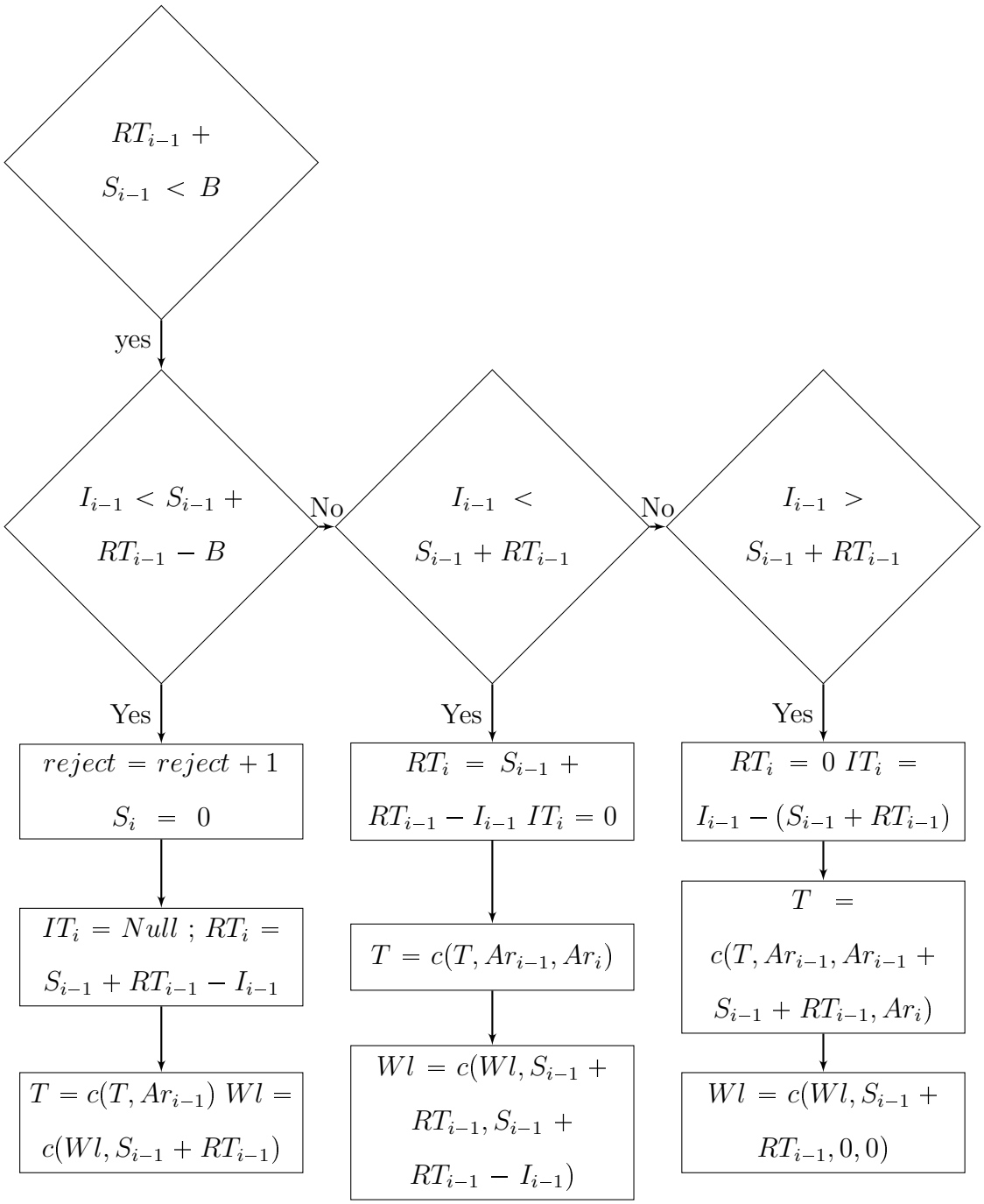
(a) Test if IT_N is null, if yes, then

$$\begin{aligned} T &= c(T, Ar_N, Ar_N + RT_N) \\ Wl &= c(Wl, RT_N, 0) \end{aligned}$$

(b) if no, then

$$\begin{aligned} T &= c(T, Ar_N, Ar_N + RT_N + S_N) \\ Wl &= c(Wl, RT_N + S_N, 0) \end{aligned}$$

(5) Revise (d) of Step 4 in Algorithm 3.1:



This algorithm would give us the desirable result in terms of allowing full access to service despite the bound. Further, we are able to get frequency function of the workload.

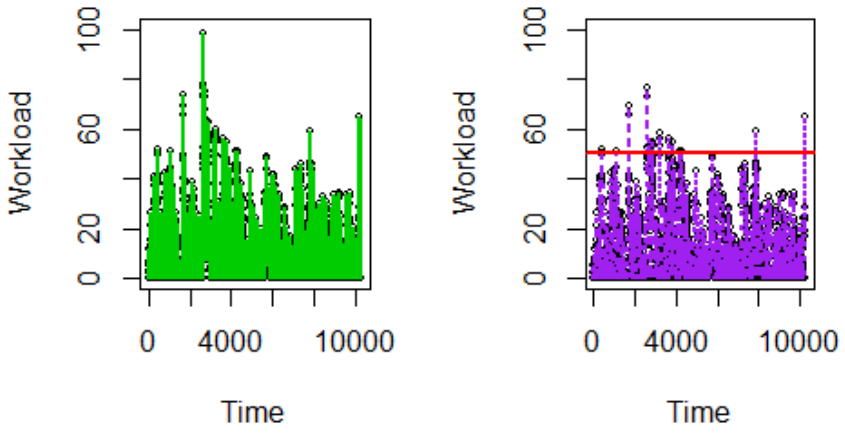


FIGURE 4.7. Workload vs Timel

Figure 4.7 shows Left: Workload without a bound, same as the left graph of Figure 3.3.1. Right: Workload with a bound at 50 but full access to service is allowed using same wait and service dataset.

This method of allowing full access does not help increase the proportion of idle time for the server as 36.3% of time is idle comparing with 36.0% of time in the case without a bound. And the sever will finish its job at the same time for both cases at 10291.54 units of time. But only 6 customers are rejected from receiving service comparing with 14 as the case in the scenario two.

Figure 4.8 shows frequency curves obtained by applying method of full access to service in cases where different bounds of workload are imposed using the same dataset of inter-arrival and service as in Figure 4.7.

The problem of this method is that if we set workload bound too high, it may not make much difference in helping reduce overload. As blue and green curves in Figure 4.8 show, although we set the bound of workload at 40 and 50 respectively as in Figure 4.5, their frequency curves are very close to the one without a bound, which result in a nearly zero improvement in reducing overload. On the other hand, if we set bound at 30, it does help reduce the frequency of overload. In

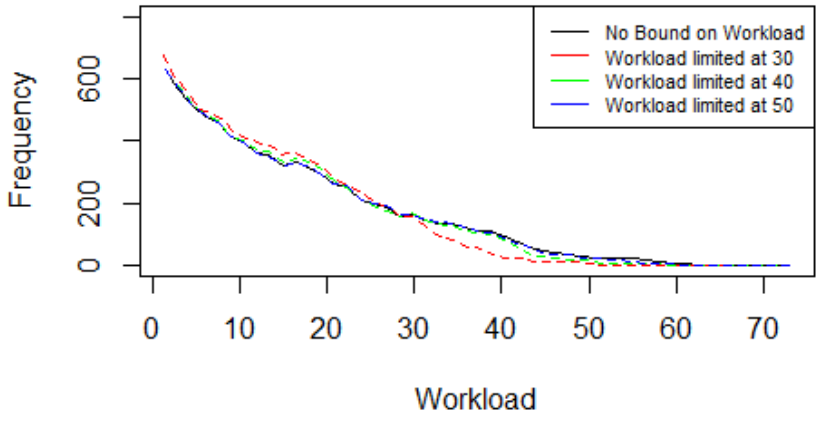


FIGURE 4.8. Frequency workload in Full Access model

other word, in order to reduce overload, one has to set the bound lower than what is acceptable, 50 in this case. But this would cause another problem that we may have to reject more customers from receiving their service for the purpose of reducing overload.

CHAPTER 5

PDF of Workload with Bound

5.1. Constructing A PDF of Workload

We have obtained frequency curves with imposed bound using three different methods discussed in the previous chapter. Now we would like to construct a pdf of workload with application of these methods. As a matter of fact, in reality, it is difficult to obtain the pdf of workload even given the condition that the interarrival and service time follow particular distributions. However, we can approximate the pdf of workload using the frequency function of workload and then convert it into a pdf. We can simply divide the frequency value at every point of workload by the area underneath the frequency curve, which would give us an approximation of the pdf of workload.

However, in previous frequency graphs we obtained, the particular frequency at workload of zero is not included in these curves. That is because when workload is at 0, the server is in an idle state. We cannot count the frequency at workload 0 over the whole period because workload at 0 is a discrete value that has its own probability, and the distribution of workload is mixed.

$$P(W = 0) = \pi_0 > 0$$

Workload has a partial probability density function $f(W)$ when it is greater than 0 which accounts for a total probability of $(1 - \pi_0)$. Therefore, we will have a probability function of workload as follows

$$P(W) = \begin{cases} f(W) & W > 0 \\ \pi_0 & W = 0 \end{cases}$$

In order to get an approximation of the pdf of workload, we first look to obtain $\hat{\pi}_0$, the estimate of π_0. This is easy to get that since we have information about the idle time for each customer, IT_i, and the total period max(T), so we can calculate $\hat{\pi}_0$ by the proportion of idle time from the simulation

$$\hat{\pi}_0 = \frac{\sum_{i=1}^{\infty} IT_i}{max(T)}$$

and then divide the frequency curve of workload we obtained before, denote here as $Frq(W)$. Hence the estimate of partial pdf of workload when it is greater than 0 is

$$\hat{f}(W) = (1 - \hat{\pi}_0) * Frq(W)/auc \quad (W > 0)$$

Here auc denotes the area under the frequency curve obtain by

$$auc = \int_{0^+}^{\infty} Frq(W)dW$$

Now we have a partial pdf of workload when workload is greater than 0 such that

$$\int_{0^+}^{\infty} \hat{f}(W)dW = (1 - \hat{\pi}_0)$$

Thus, our estimated pdf of workload is

$$\hat{P}(W) = \begin{cases} (1 - \hat{\pi}_0) * Frq(W)/auc & W > 0 \\ \hat{\pi}_0 & W = 0 \end{cases}$$

5.2. An example of building pdf of workload in case of partial access

To illustrate, we would like to obtain a pdf of workload while a bound value of 50 is imposed. We use the same system with $\mu = 0.15$ and $\lambda = 0.1$ and only allow a partial access to service as discussed in the previous chapter.

5.2.1. Obtain the Probability of Idle Time. In order to obtain a more accurate estimate of the pdf of workload, first we need to obtain the probability of idle time π_0. One way to achieve that is by simulating as many customers as

possible in a sense that the proportion of idle time will be approaching to the true probability of idle time. Since both idle time IT and the total period T is subject to the change with the number of customers C, we can consider these two quantities as functions of C. Hence once we increase C to infinity, the estimate $\hat{\pi}_0$ should approach π_0:

$$\lim_{C\to\infty} \sum_{i=1}^{C} IT_i/T(C) \approx \pi_0$$

In addition to boosting the number of customers, we also adopt the method of resampling. We can resample N data from the collected N data of interarrival and service time and generate a set of resampled data: (I^1, S^1), (I^2, S^2) ... (I^R, S^R). For any set of (I^k, S^k), we are able to get an estimate $\hat{\pi_0}^k$, $k = 1, 2 \ldots R$. Hence, the average of $\hat{\pi_0}^k$ will provide a more reliable estimate of π_0. What's more, if we continue this resampling procedures for a large number of times, our averaged estimate should be closing to the true probability of idle time π as well:

$$\lim_{R\to\infty} \sum_{k=1}^{R} \hat{\pi_0}^k/R \approx \pi_0$$

However, once we adopt either method or both, it is inevitable to encounter computational problems caused by the increased volume of data. Our goal is to get a reliable estimate of π_0 without sacrificing too much time on computing the result.

Figure 5.1 shows a heat map of squared residuals, on the x-axis is the number of customers and y-axis stands for the resampling repeats.

Figure 5.1 sheds light on how the estimate of π_0 varies by applying resampling procedures R and adjusting the number of customers C. Each of this 5 $\times$ 5 grid represents a combination of resampling repeats and number of customers, which leads to an estimate of π_0, $\hat{\pi}_0(ij)$. In this simulation context, we are able to obtain the true probability of idle time by specifying C large enough, say 100,000, so we the true probability $\pi_0 = 0.3568$. The squared residual grid $(\hat{\pi}_0(ij) - \pi_0)^2$ $(i, j = 1, 2, 3, 4, 5)$ would provide a good measure of goodness of fit. Then the grid of

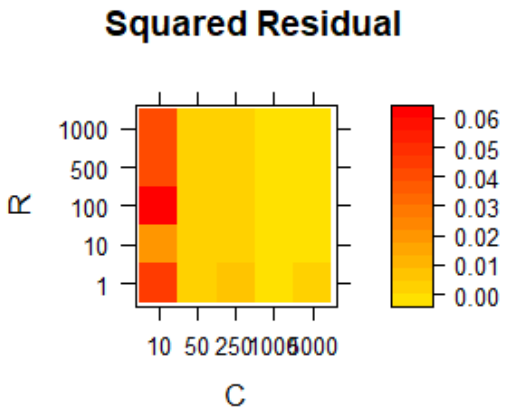


FIGURE 5.1. Resampling

squared residual can be converted into a heat map based on numeric values as shown in Figure 5.1.

The above graph shows that, in order to obtain an accurate estimate of π_0, one does not have to generate a large set of data or implement a large scale of resampling procedures. The combination of 50 customers and 10 times of resampling would produce nearly a perfect estimate of 0.36.

5.2.2. Obtain the Partial PDF of Workload. Now, as we have obtained the probability of idle time $\hat{\pi_o}$, we can assign a total probability of $(1 - \hat{\pi_0})$ to the partial pdf of workload. Then we use one of sets (I^k, S^k) to obtain a frequency graph as shown on the left of Figure 3.4.1. Also, the area under the curve is shadowed with grey color and we can obtain the area by dividing the whole period into 500 equal pieces with $x = (0.1, 0.2, ..., 50)$, so the area would be

$$auc = 0.1 * \sum_{i=1}^{499} Frq(x_i)$$

Finally,

$$\hat{f}(W) = (1 - \hat{\pi}_0) * Frq(W)/auc \quad (W > 0)$$

so we can construct a partial pdf of workload shown on the right of Figure 4.1.1.

Figure 5.2 shows: (Left) Frequency curve obtained by applying method of partial access, (Right) Resulting PDF of the workload.

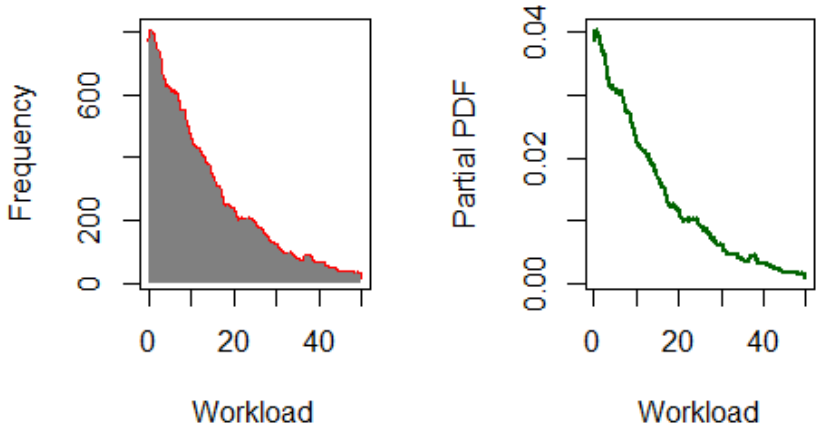


FIGURE 5.2. Frequency curve and pdf

We can see the partial pdf $\hat{f}(W)$ has the same shape as the frequency curve, and we also estimate π_0 by 0.36. So our approximation to the distribution of workload is given by

$$\hat{P}(W) = \begin{cases} 0.36 & W = 0 \\ \hat{f}(W) & 0 < W \leqslant 50 \\ 0 & W > 50 \end{cases}$$

in which $\hat{f}(W)$ is given by the partial PDF on the right plot of Figure 5.2.

5.2.3. Improvement to the PDF Using Resampling. In section 5.2.2, we have obtained an approximated mixed distribution of workload, which is useful in terms of understanding a server's workload under a bounded queueing system. However, as the pdf is shown in the previous section, it is noticeable that the partial pdf of workload takes a bumpy shape. This may cause a problem. For example, if one wants to know the true probability of workload within a certain range, say from 20 to 30, it could give us a bad result (probability) using this partial pdf since the pdf is too flexible such that any prediction from this pdf would have a high variability. In fact, as Figure 5.3 shows, if we draw frequency

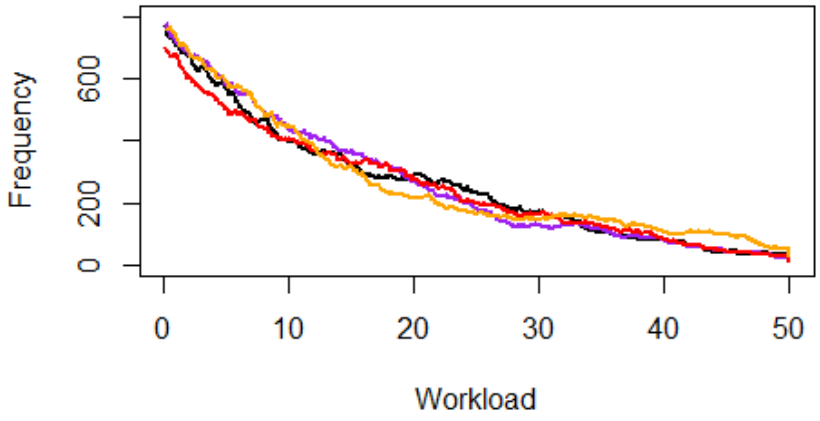


FIGURE 5.3. Different Frequency curves

curves based on different sets of (I^k, S^k), we may end up getting curves that are quite different.

Figure 5.3 shows frequency curves using four datasets (I^k, S^k), $k = 1, 2, 3, 4$.

In Figure 5.3, it is clear that despite using the same system with same parameters, we still get different looks of frequency curves, which are used to translate into the pdf of workload we want. Hence, the difference of frequency curve would lead to a variety of pdf's, and eventually result in a highly variable estimate of the true partial pdf.

Ideally, we would like to have a smooth partial pdf to avoid high variability and to have a better estimation of the true distribution. In order to achieve that, we turn to resampling to give us a better estimate of the target pdf.

The method is easy to interpret, it is as follows:

(1) Specify queueing system as M/M/1 and its inter-arrival λ and service rates μ as well as the number of customer N

(2) Set a bound value B

(3) Randomly generate K datasets of inter-arrival and service time

(4) Obtain idle time $IT_i, i = 1, 2, ...N$ and total period T for each individual datasets, calculate $\hat{\pi}_0^1$, $\hat{\pi}_0^2$, ... $\hat{\pi}_0^K$

(5) Build a Frequency function based on individual dataset: $Frq_1(W)$, $Frq_2(W)$, ... $Frq_N(W)$.

(6) Create an ordered set of workload $X = (x_1, x_2, ...x_M)$ where $M = 10 * B$ in which every adjacent elements has a difference of 0.1

(7) Build a matrix $P = [Frq_1(X), Frq_2(X), ..., Frq_K(X)]_{M*K}$ that takes a form

$$\begin{array}{c} \begin{array}{cccc} Frq_1 & Frq_2 & ... & Frq_K \end{array} \\ \begin{pmatrix} p_{11} & p_{12} & ... & p_{1K} \\ p_{21} & p_{22} & ... & p_{2K} \\ . & . & ... & . \\ . & . & ... & . \\ . & . & ... & . \\ p_{M1} & p_{M2} & ... & p_{MK} \end{pmatrix} \end{array} \begin{array}{c} x_1 \\ x_2 \\ . \\ . \\ . \\ x_M \end{array}$$

(8) Obtain the mean value of each row from the matrix P and standard deviation,

$$P_i = \frac{1}{K} \sum_{j=1}^{K} p_{ij} \quad (i = 1, 2, ...M)$$

$$Sd_i = \frac{1}{K-1} \sqrt{\sum_{j=1}^{K} (p_{ij} - P_i)^2} \quad (i = 1, 2, ...M)$$

and the mean of $\hat{\pi}_o^i$,

$$\hat{\pi}_o^* = \frac{1}{K} \sum_{j=1}^{K} \hat{\pi}_o^i$$

(9) Fit a frequency curve Frq^* using set X as workload (x-axis) and P_i as corresponding frequency.

(10) Calculate the area under the curve Frq^*, auc

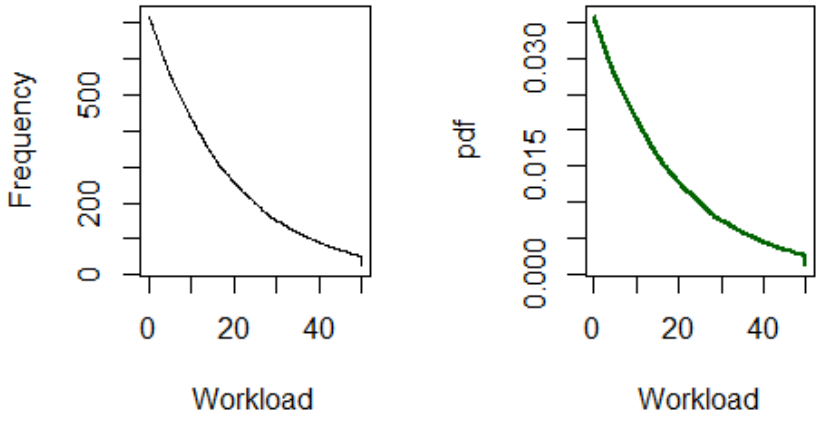


FIGURE 5.4. Smoothed Frequency and PDF

(11) The approximation to the pdf of workload would be

$$\hat{P}(W) = \begin{cases} \hat{\pi}_o^* & W = 0 \\ (1 - \hat{\pi}_o^*) Frq^*/auc & 0 < W \leqslant B \\ 0 & W > B \end{cases}$$

Now, we use the procedures discussed above to obtain an averaged estimate of the pdf of workload for the example in section 3.4.

Figure 5.4 shows (Left) the averaged frequency curve, (Right) the pdf derived from the averaged frequency curve.

Here, we specify $K = B = 50$, and the number of customer $N = 1000$. Using the same system as the example in 3.4, we obtain a frequency curve on the left of Figure 3.5.2 and the following partial pdf of workload.

After completing such a procedure, the estimate of partial pdf of workload becomes smoother and less variable.

Also, we have a set of estimates of idle probability $\hat{\pi}_0^i$, and by averaging them, we will get a more stable estimator of idle probability. In this particular case,

we have a mean of 0.358. Therefore, the estimated partial pdf of workload takes 64.2% probability when the server is busy.

CHAPTER 6

Conclusion

This paper is mainly concerned with presenting the behaviour of workload using simulated datasets. Although in reality there would be an inevitable discrepancy with our simulated results (caused by the disparity between the actual and the assumed distributions of inter-arrival and service time), they provide us with general ideas of the relationship of workload with time and frequency. For example, if we simulate our interarrival and service dataset using the M/M/1 model, we notice the corresponding partial pdf of workload takes a form of an exponential distribution as well. In situations of imposing limits on workload, different methods to tackle it would lead to different workload-time graphs, through which we are able to find the best form based on our needs such as "finishing serving all customers fastest", "having the least proportion of idle time", "serving the maximum number of customers within a certain period," etc.

As in the book [4] by Brill (2017), we can build a relationship between upcrossing rate and downcrossing rate. By equating them and solving the resulting equation, we are able to obtain the distribution of workload. However, aside from complex mathematical operations involved in figuring out the distribution, another prerequisite for this equation to hold is that limits should not be imposed on workload. In other word, if there exists a bound on workload and it is small enough to effectively impact the workload (If bound value is too high, it will have limited effect on workload), this up-and-down crossing equation is difficult to study.

On the other hand, since we have found a way to present the workload-time graph under this workload-bound restriction in a context of large dataset and translate it into a pdf, we are able to avoid difficulties involved in obtaining the pdf by mathematical operations and straightforwardly get an approximation of the distribution. However, in order for our pdf to be a good approximation, we

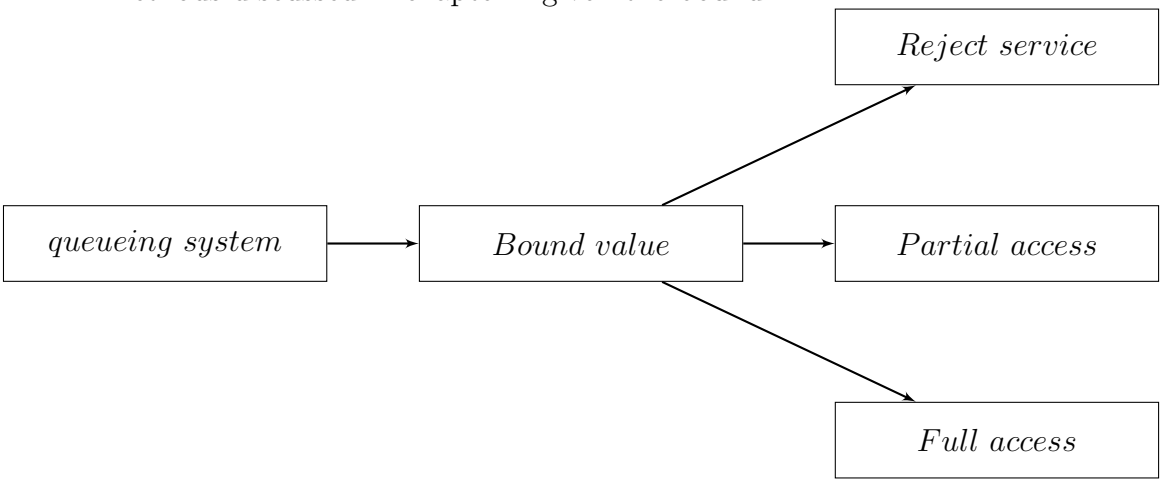
are also required to simulate a dataset large enough to truly observe patterns and apply techniques such as resampling at the same time.

Future Work

Thus far, we have considered applying those three methods to deal with a workload bound restriction when the dataset of inter-arrival and service time are fixed, so does the bound value. However, there is an interesting problem concerning the bound value. Suppose we have a restaurant (queueing system), and its owner wants to make money by having a lot of customers in a day. But the restaurant has only one server, and the server needs a break during the work. Suppose the restaurant is always busy and customers keep coming one by one. There has to be a workload limit for the server. We also know the distribution of customer interarrival time as well as service time. So our goal will be to figure out a way to meet both criteria of the owner and the server.

- Owner's perspective: serve as many customers as possible in a day
- Server's perspective: get as much break time as possible

Now we may set that the desired number of customers for the owner is n_o and the desired proportion of idle time for the server is p_s. In order to achieve goals of both sides, we need to adjust the bound value of workload and apply those three methods discussed in chapter 4 given the bound.



Possible future work might be to make decisions on what method we should apply and under what circumstances.

Bibliography

[1] U. Bhat. (2008) An Introduction to Queueing Theory: Modeling and Analysis in Applications. Springer.

[2] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi. (2006) Queueing Networks and Markov Chains. (2nd ed.) Wiley.

[3] P.H. Brill. (2000), P.H., A Brief Outline of the Level Crossing Method in Stochastic Models, Canadian Operational Research Society Newsletter.

[4] P.H. Brill, Percy H. (2017) Level Crossing Methods in Stochastic Models, 2nd edition. Springer.

[5] R.B. Cooper. (1981) Introduction to Queueing Theory. North Holland Publishers.

[6] N. Gautam. (2012) Analysis of Queues. CRC Press.

[7] D. Gross, J. Shortle, J. Thompson, C. Harris. (2008) Fundamentals of Queueing Theory. 4th edition. Wiley.

[8] J. Jain, S.G. Mohanty, W. Bohm. (2007) A Course on Queueing Models. CRC Press

[9] L. Kleinrock. (1975) Queueing Systems. Volume 1. John Wiley.

[10] J. Medhi. (2002) Stochastic Models in Queueing Theory. 2nd edition. Academic Press.

[11] L. Takacs. (1962) Introduction to the Theory of Queues. Oxford University Press.

Appendix: R Code

```
# Specify C and B and randomly generate interarrival and service time


N = 1000        % Number of customers
TH = 50             % Maximum Threshold of Workload

set.seed(11)
OI <- rexp(N - 1,0.1)   % interarrival time for 1000 customers
OS <- rexp(N, 0.15)  % service time for 1000 customers

I <- sample(OI, N-1, replace = TRUE)
S <- sample(OS, N, replace = TRUE)

Pi = 1-(0.1/0.15)
T <- 0
WLT <- 0
g <- rep(0,N)
W <- rep(0,N)

Ar=0
cumul=0
n=1
while (n < N) {
  cumul <- cumul + I[n]
  Ar <- c(Ar,cumul)
  n = n + 1
}


# PAS algorithm


i = 2

while (i < N + 1) {

  if(g[i-1] + S[i-1] < TH){

      if(g[i-1] + S[i-1] < I[i-1]){

        g[i] = 0
        W[i] = I[i-1] - (g[i-1] + S[i-1])
        T <- c(T, Ar[i-1], Ar[i-1] + g[i-1] + S[i-1], Ar[i])
        WLT <- c(WLT, g[i-1] + S[i-1], 0, 0)
      }
      else{
```

```
      g[i] = g[i-1] + S[i-1] - I[i-1]
      W[i] = 0
      T <- c(T, Ar[i-1], Ar[i])
      WLT <- c(WLT, g[i-1] + S[i-1], g[i])
    }
  }
  else{

    if(TH < I[i-1]){

      g[i] = 0
      W[i] = I[i-1] - TH
      T <- c(T, Ar[i-1], Ar[i-1] + TH, Ar[i])
      WLT <- c(WLT, TH, 0, 0)
    }
    else{

      g[i] = TH - I[i-1]
      W[i] = 0
      T <- c(T, Ar[i-1], Ar[i])
      WLT <- c(WLT, TH, g[i])
    }
  }

  i = i + 1

}
sum(W)/max(Ar)
T <- c(T, Ar[N], Ar[N] + min(TH, g[N] + S[N]))
WLT <- c(WLT, min(TH, g[N] + S[N]), 0)


# RS algorithm


i = 2
reject = 0
while (i < N + 1) {

  if(g[i-1] + S[i-1] < TH){

    if(g[i-1] + S[i-1] < I[i-1]){

      g[i] = 0
      W[i] = I[i-1] - (g[i-1] + S[i-1])
      T <- c(T, Ar[i-1], Ar[i-1] + g[i-1] + S[i-1], Ar[i])
      WLT <- c(WLT, g[i-1] + S[i-1], 0, 0)
    }
```

```
    else{

      g[i] = g[i-1] + S[i-1] - I[i-1]
      W[i] = 0
      T <- c(T, Ar[i-1], Ar[i])
      WLT <- c(WLT, g[i-1] + S[i-1], g[i])
    }
  }
  else{

    if(g[i-1] < I[i-1]){

      reject = reject + 1
      g[i] = 0
      W[i] = I[i-1] - g[i-1]
      T <- c(T, Ar[i-1] + g[i-1], Ar[i])
      WLT <- c(WLT, 0, 0)
    }
    else{

      reject = reject + 1
      g[i] = g[i-1] - I[i-1]
      W[i] = 0
      T <- c(T, Ar[i])
      WLT <- c(WLT, g[i])
    }
  }
  i = i + 1
}

if(g[N]+S[N]>TH){
  reject = reject + 1
  T <- c(T, Ar[N] + g[N])
  WLT <- c(WLT, 0)
}else{
  T <- c(T, Ar[N], Ar[N] + g[N] + S[N])
  WLT <- c(WLT, g[N] + S[N], 0)
}


# FAS algorithm


i = 2
reject = 0

while (i < N + 1) {

  if(g[i-1] + S[i-1] < TH){
```

```
    if(g[i-1] + S[i-1] < I[i-1]){

      g[i] = 0
      W[i] = I[i-1] - (g[i-1] + S[i-1])
      T <- c(T, Ar[i-1], Ar[i-1] + g[i-1] + S[i-1], Ar[i])
      WLT <- c(WLT, g[i-1] + S[i-1], 0, 0)
    }
    else{

      g[i] = g[i-1] + S[i-1] - I[i-1]
      W[i] = 0
      T <- c(T, Ar[i-1], Ar[i])
      WLT <- c(WLT, g[i-1] + S[i-1], g[i])
    }
  }
  else{

    if(I[i-1] < S[i-1] + g[i-1] - TH){

      reject = reject + 1
      g[i] = S[i-1] + g[i-1] - I[i-1]
      S[i] = 0
      W[i] = NA
      T = c(T, Ar[i-1])
      WLT = c(WLT, S[i-1] + g[i-1])
    }
    else if(I[i-1] < S[i-1] + g[i-1]) {

      g[i] = S[i-1] + g[i-1] - I[i-1]
      W[i] = 0
      T <- c(T, Ar[i-1], Ar[i])
      WLT <- c(WLT, S[i-1] + g[i-1], S[i-1] + g[i-1] - I[i-1])
    }
    else{
      g[i] = 0
      W[i] = I[i-1] - (S[i-1] + g[i-1])
      T = c(T, Ar[i-1], Ar[i-1] + g[i-1] + S[i-1], Ar[i])
      WLT = c(WLT, S[i-1] + g[i-1], 0, 0)
    }
  }

  i = i + 1
}
sum(W)/max(Ar)
if(is.na(W[N])){
  T <- c(T, Ar[N], Ar[N] + g[N])
  WLT <- c(WLT, g[N], 0)
}else{
```

```
  T <- c(T, Ar[N], Ar[N] + g[N] + S[N])
  WLT <- c(WLT, g[N] + S[N], 0)
}


# Frequency function


f = 0

frq <- function(workload){

  i = 1

  while (T[i] < max(T)) {

    if(T[i] == T[i+1]){

      if((WLT[i] <= workload) & (workload < WLT[i+1])) {f = f + 1}
    }

    else {

      if((WLT[i+1] < workload) & (workload <= WLT[i])) {f = f + 1}
    }

    i = i + 1
  }

  return(f)
}

Frequency <- Vectorize(frq)


# Resampling


dev <- matrix(data = 0, nrow = 5, ncol = 5)
r <- c(1, 10, 100, 500, 1000)

N = 5000        # Number of customers
TH = 50             # Maximum Threshold of Workload

set.seed(11)
OI <- rexp(N - 1,0.1)    # interarrival time for 1000 customers
OS <- rexp(N, 0.15)  # service time for 1000 customers

for (k in 1:5) {
```

```
sumpi = 0

for (j in 1:r[k]) {

  I <- sample(OI, N-1, replace = TRUE)
  S <- sample(OS, N, replace = TRUE)

  T <- 0
  WLT <- 0
  g <- rep(0,N)
  W <- rep(0,N)

  Ar=0
  cumul=0
  n=1
  while (n < N) {
    cumul <- cumul + I[n]
    Ar <- c(Ar,cumul)
    n = n + 1
  }

  i = 2

  while (i < N + 1) {

    if(g[i-1] + S[i-1] < TH){

      if(g[i-1] + S[i-1] < I[i-1]){

        g[i] = 0
        W[i] = I[i-1] - (g[i-1] + S[i-1])
        T <- c(T, Ar[i-1], Ar[i-1] + g[i-1] + S[i-1], Ar[i])
        WLT <- c(WLT, g[i-1] + S[i-1], 0, 0)
      }
      else{

        g[i] = g[i-1] + S[i-1] - I[i-1]
        W[i] = 0
        T <- c(T, Ar[i-1], Ar[i])
        WLT <- c(WLT, g[i-1] + S[i-1], g[i])
      }
    }
    else{

      if(TH < I[i-1]){

        g[i] = 0
        W[i] = I[i-1] - TH
```

```
            T <- c(T, Ar[i-1], Ar[i-1] + TH, Ar[i])
            WLT <- c(WLT, TH, 0, 0)
          }
          else{

            g[i] = TH - I[i-1]
            W[i] = 0
            T <- c(T, Ar[i-1], Ar[i])
            WLT <- c(WLT, TH, g[i])
          }
        }

        i = i + 1

      }
      T <- c(T, Ar[N], Ar[N] + min(TH, g[N] + S[N]))
      WLT <- c(WLT, min(TH, g[N] + S[N]), 0)

      sumpi = sumpi + sum(W)/max(T)

    }

    dev[5,k] = sumpi/r[k]
}

library(lattice)
colnames(dev) = c(1,10,100,500,1000)
rownames(dev) = c(10,50,250,1000,5000)
levelplot((dev1-0.3568534)^2, data=data, xlab="C", ylab = "R",
          col.regions = heat.colors(150)[length(heat.colors(100)):1],
          main="Squared Residual")
```

Vita Auctoris

Zhanxuan Ding was born in 1994 in Zhuji, Zhejiang, China. He completed his Bachelor of Science degree in Shenyang Ligong University, Shenyang, China in 2017. He is currently a candidate for Master of Science degree in Statistics at University of Windsor (Windsor, ON) and he is expected to graduate in Fall 2018.