

COMBINING SOME COMPUTER SYSTEM PERFORMANCE ANALYSIS WITH ALGORITHM AND OBJECT DEVELOPMENT IN TEACHING

Dale R. Fox

Quinnipiac University
Dale.zy.fox@quinnipiac.edu

Introduction

Developing at least some elementary models that can be used to help decision makers in their computer network design is important. For example, at almost the simplest levels, both authors have been on university committees that were concerned about the utilizations of T1 and/or fractional T3 lines for campus internet access. These same sorts of issues arise in businesses. There are an abundance of situations where having models to help decision makers is important. Some examples of these situations include, selecting access line, router, switch and server processing capabilities so that they contribute to overall network performance.

The types of interactions that occur in computer networks can be quite complicated. The processing times and demands almost surely need to be modeled using probability distributions in order to account for the variability involved. The interconnectivity and topology of a network also impact overall performance. But rather than get involved in the more sophisticated network models we will make use of one of the most well known queueing models in order to motivate analysis of system performance and develop the GUI, objects/classes, algorithms and code in order to make use of the model.

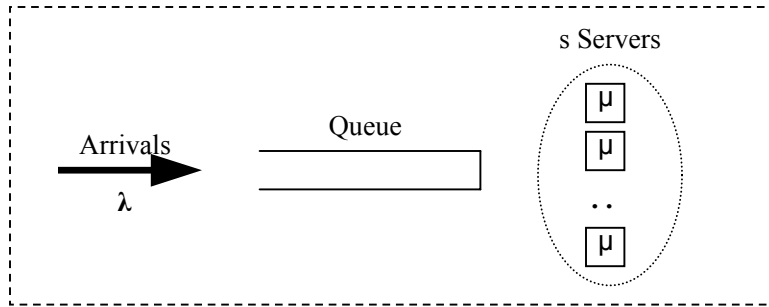
The Queueing Model

The model we will consider is often called the M/M/s queueing system since

- the distribution of time between arrivals is exponential
- the distribution of the processing time at each server is exponential
- there are s different servers

It is important to not confuse the queueing theory use of the word server with the computer use of the word. Servers in queueing models are anything that provide service, which may well be trunk lines, multiplexers, routers, switches or servers in computer networks, to name a few possibilities.

We present the model at a very general level of description since there are many references on the subject. The following is a typical diagram used to represent the system.



where

λ = average arrival rate

μ = average service rate for each server

S = the number of servers

These are the inputs that must be measured in order to perform any analysis.

We display the solutions for the steady state probabilities to save the reader from searching them out and likely having to adjust the notation.

In order for the system to be stable the following condition must be satisfied.

$$utilization = \rho = \lambda / s\mu < 1$$

If this condition is satisfied then the following are the equations for the steady state probabilities.

$\pi(n)$ can be interpreted as the overall percentage of time there are exactly n items in the system. So

$$\pi(n) = \begin{cases} \frac{\left(\frac{\lambda}{\mu}\right)^n}{n!} \pi(0) & \text{for } n = 1, 2, \dots, s \\ \frac{\left(\frac{\lambda}{\mu}\right)^n}{s! s^{n-s}} \pi(0) & \text{for } n = s + 1, s + 2, \dots \end{cases}$$

where

$$\pi(0) = \left[\left(\sum_{n=0}^{s-1} \frac{\left(\frac{\lambda}{\mu}\right)^n}{n!} \right) + \frac{\left(\frac{\lambda}{\mu}\right)^s}{s!} \frac{1}{1 - \left(\frac{\lambda}{s\mu}\right)} \right]^{-1}$$

Substituting the expression for the utilization into this equation gives the following.

$$\pi(0) = \left[\left(\sum_{n=0}^{s-1} \frac{\left(\frac{\lambda}{\mu}\right)^n}{n!} \right) + \frac{\left(\frac{\lambda}{\mu}\right)^s}{s!} \frac{1}{(1-\rho)} \right]^{-1}$$

Using a generating function technique to derive L_q , the expected performance measures can be shown to be

$$L_q = \frac{\pi(0) \left(\frac{\lambda}{\mu}\right)^s \rho}{s!(1-\rho)^2} \quad W_q = \frac{L_q}{\lambda}$$

$$W = W_q + \frac{1}{\mu} \quad L = \lambda W = L_q + \frac{\lambda}{\mu}$$

where

L_q = The expected number in the queue (waiting for service)

L = The expected number in the system (in the queue and in service)

W_q = The expected time in the queue (waiting for service)

W = The expected time in the system (in the queue and in service)

Notice that all of these expressions are computed purely on the inputs of λ , μ , s and $\pi(0)$.

It probably is safe to assume that all of these formulas seem very complicated to those that are unfamiliar with them. This is even likelier to be true of students. Yet these are useful formulas that should help motivate students to solve them on a computer rather than by hand. They are also not the standard examples used for teaching algorithmic effectiveness, some of which can be quite abstract to students. In addition, developing a reasonable GUI is important to ease implementation and interpretation of solutions.

So how can we improve the computational effectiveness?

Improving Computations

For students, it seems that two of the biggest things that need to be noticed about these formulas are

- some things should be computed before others
- some things are slight adjustments of other things

These may not be immediately obvious, but they are likely to be important whenever someone wants generate solutions using a computer.

Possibly, most important,

- all of the other performance measures depend on $\pi(0)$.
 - this is true for the probabilities and expectations
- calculating $\pi(0)$ depends only on the inputs λ , μ and s

In a similar vein, for computing the expectations, once L_q has been calculated the remaining expectations, L , W_q and W can be determined purely from L_q and the three inputs λ , μ and s .

These observations will definitely determine the order of our calculations.

Calculating $\pi(0)$

$\pi(0)$ can be considered to be a normalizing constant that ensures the steady state probabilities sum to one. Since there are an infinite number of potential states of the system, part of the denominator is actually determined from a closed form solution for an infinite sum.

$$\pi(0) = \left[\left(\sum_{n=0}^{s-1} \frac{\left(\frac{\lambda}{\mu}\right)^n}{n!} \right) + \frac{\left(\frac{\lambda}{\mu}\right)^s}{s!} \frac{1}{(1-\rho)} \right]^{-1}$$

finite sum
inf inite sum

We can calculate the finite sum iteratively. Notice that the upper limit is one less than the number of servers. Each term in the finite sum is

$$n^{\text{th}} \text{ term} = \left(\frac{\lambda}{n\mu}\right) (n-1)^{\text{st}} \text{ term}$$

So this is an easy computation where the sum of (s-1) terms is accumulated within a loop.

Fortunately, the infinite sum term can be developed by taking the iteration for the finite sum one more step without taking the sum since

$$s^{\text{th}} \text{ term} = \left(\frac{\lambda}{s\mu}\right) (s-1)^{\text{st}} \text{ term}$$

Then this needs to be multiplied times

$$\frac{1}{(1-\rho)} = \frac{s\mu}{s\mu - \lambda}$$

to get the infinite sum portion.

Then these two portions are added together and raised to the -1 power to get the fraction.

Calculating $\pi(n)$

A similar sort of iteration can be developed for the $\pi(n)$ s. We restate the equation for the steady-state probabilities in an iterative form.

$$\pi(n) = \begin{cases} \left(\frac{\lambda}{\mu}\right)^n \pi(0) & \text{for } n = 0, 1, 2, \dots, s \\ \left(\frac{\lambda}{\mu}\right)^s \pi(s) & \text{for } n = s + 1, s + 2, \dots \end{cases}$$

This really is the essentially the same iteration used to generate the terms in the finite sum except once $n > s$ the denominator stays at s . This is because there are only s servers and the overall system service rate can't increase beyond this.

But it is also the case that this iterative scheme starts with knowledge of $\pi(0)$. So this must be computed first.

Calculating Expectations

As noted before, the expectations are all configured as functions that can be developed from L_q and the three inputs λ , μ and s . L_q is purely a function of the three inputs λ , μ and s .

$$L_q = \frac{\pi(0) \left(\frac{\lambda}{\mu}\right)^s \rho}{s!(1-\rho)^2}$$

There are no summations. Notice the term

$$\frac{\left(\frac{\lambda}{\mu}\right)^s}{s!}$$

is the only term requiring much iterative computation and we already calculated it when calculating the infinite sum portion of $\pi(0)$.

The Code

We have a rather elaborate set of code configured to run as an applet at the following link.

<http://cisdev.quinnipiac.edu/fox/javaApplets/MMSApplet.html>

This should run on a fairly recent browser without any special plugins. The code is available on request.

The code has been configured to compute these performance measures in methods, passing and returning what is important. They illustrate the computational improvements discussed earlier in this paper.

There are several features of special interest in teaching and for general use.

- The input validation is done in a method making use of try and catch block in addition to other approaches to accumulate an appropriate error message. This method returns a boolean which is used to prevent or invoke computation of other parts of the program.

- There is a separate class for the RadioButtonHandler so that the GUI responds as the user selects a radio button
- There is a separate class for the SliderHandler so tha the GUI responds as the user moves the slider
- Distinct methods are used to develop different parts of the GUI which return JPanel pieces to be assembled

Some of the less important and more easily implemented features are

- Using a GridBagLayout to construct each JPanel and then putting the panels together with a GridLayout
- A special method has been developed so that the gridBagConstraints can be developed with more clarity
- Only the input JTextFields are editable

References

Deitel and Deitel, Java: How to Program, 4th Edition, Prentice Hall, 2002.

Gutz, Up To Speed with Swing, 2nd Edition, Manning, 2000.

Kleinrock, Queuing Systems, Volume I: Theory, Wiley Interscience, 1975.

LeMay and Cadenhead, Java 2 in 21 Days, SAMS, 2000.

Malik and Nair, Java Programming: From Problem Analysis to Program Design, Thomson Course Technology, 2003.

Sauer and Chandy, Computer Systems Performance Modeling, Prentice Hall, 1981.