Queue Simulator

Following is a study of a particular queue problem, that of a highway toll booth. A queue can be modeled as a Markov process, implying one-at-a-time exponentially distributed arrival and service times. But service times could very well have some other distribution. The following compares total time in queue and total throughput when normally distributed service times, as opposed to exponentially distributed, are assumed.

Hello Matt,

I finished the queue simulator that I mentioned in Assignment 7, problem 14, part b, and wanted to share it. If you have time to look at it and comment, I would appreciate it.

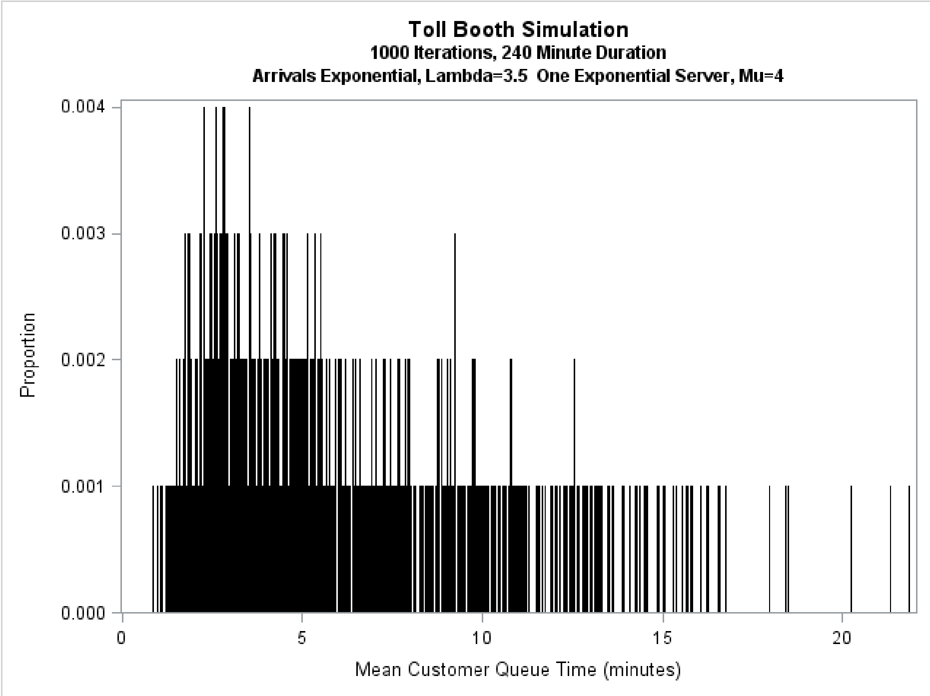
My outline is slightly different:

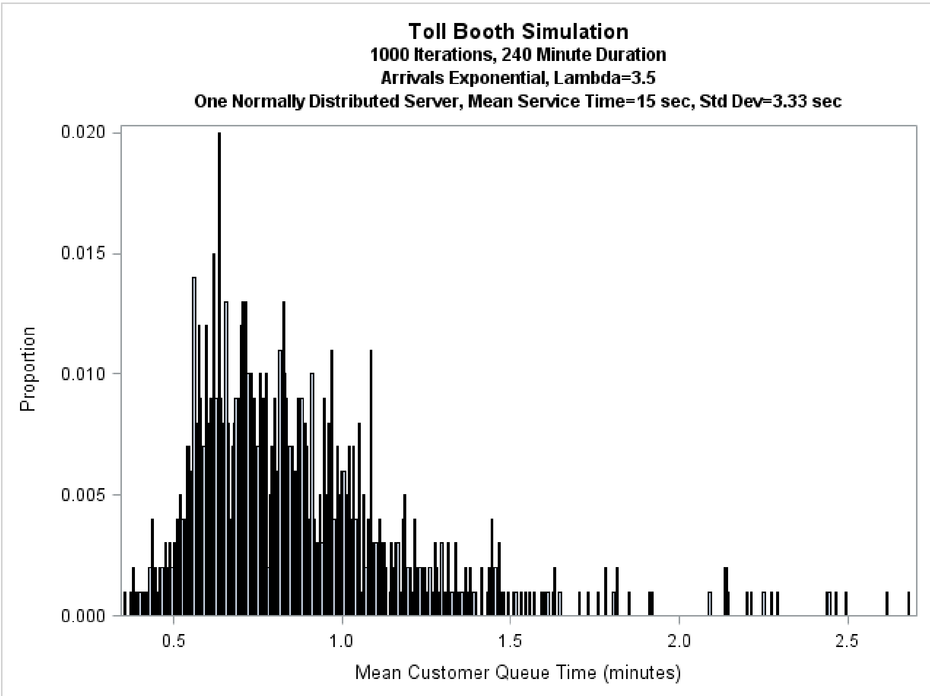
```
Outline:

1.  Initialize current epoch (to 0 minutes)

2.  Set each server status to available

3.  Randomly generate customer arrival times (from arrival probability distribution) through the
    specified end epoch
    Customers are effectively in queue but will not be recognized before the epoch reaches their arrival
    time

4.  Set current epoch to time of first arrival

5.  While time left in simulation
      While customers in queue and server available
        Assign next customer in queue (earliest arrival) to randomly selected available server
        Generate random expected service completion time (current epoch + duration) from service
        probability distribution
        Set service start time to current epoch
        Flag selected server as unavailable
        Calculate and accumulate customer's time in queue (current epoch - arrival) for analysis
      Advance epoch to minimum of next completion time of all servers and earliest arrival in queue
      While there are busy servers with completion times = current epoch
        Complete service in progress
        Flag busy servers as available
        Accumulate completions and service times by server (service time = completion time - start time)

6. Calculate final statistics:  percent service time (utilization) and queue time (average per customer)
```

The problem definition suggests that service times may not be exponentially distributed, which makes sense since servers may have idle time, at which they are not working. This interferes with the assumption that server performance, alone, predicts time to service completion

For comparison, I ran 1,000 simulations of duration of 240 minutes (4 hours) using each of one and two servers and exponential and normally distributed service times. For exponential servers, simulated mean queue times agree almost exactly with the theoretical M/M/1(2) values (1.59 minutes simulated compared to 1.75 theoretical for one server and .059 minutes (3.5 sec) simulated compared to .06 minutes (3.6 sec) theoretical for two servers). This is encouraging but, of course, M/M/1(2) assumes exponential service times.

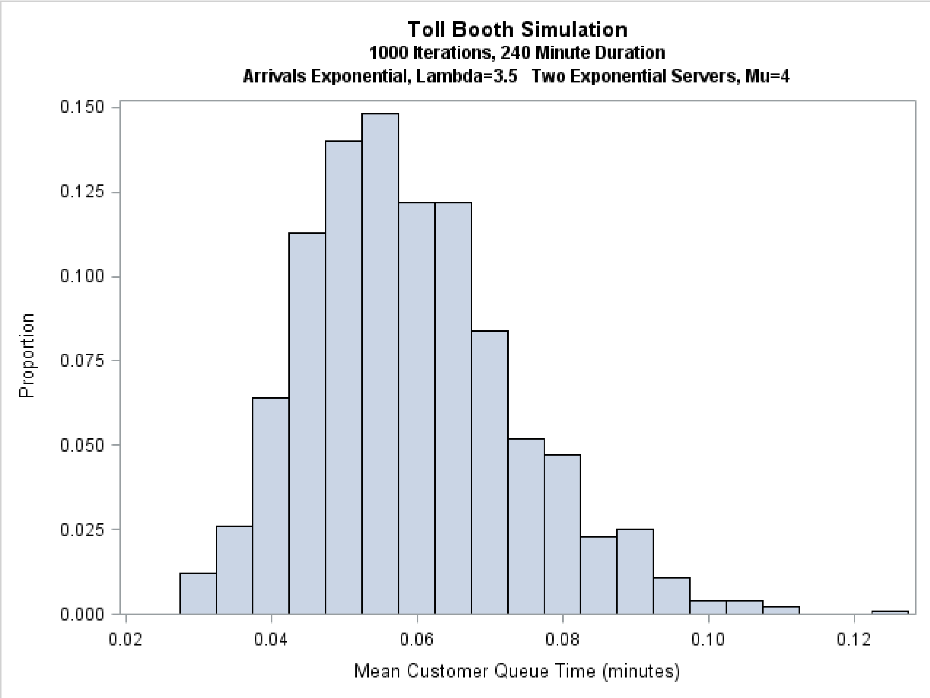
The following two histograms compare queue time distributions under single server exponential and normal service time probability distributions. Exponential mean interarrival time is 3.5 per minute, service time is 4 per minute, normal mean service time is 15 seconds (4 per minute), standard deviation is 3.33 seconds (giving an approximate 99% confidence interval of 5 to 25 seconds).

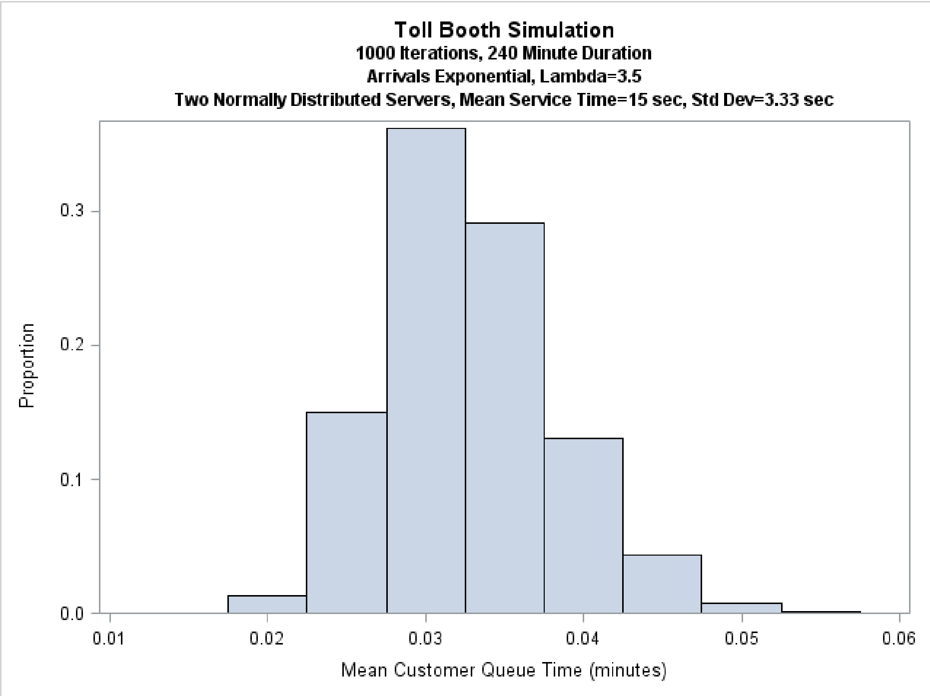




Simulated mean queue time with normally distributed service time is roughly half that of exponential service time (.88 vs. 1.59 minutes). It's interesting that both distributions exhibit a right tail (at 1,000 iterations the Central Limit Theorem hasn't yet dominated) indicating that for shorter periods, prediction models might need asymmetric compensation (lognormal, Johnson type, etc.). However, if normal service times are used (after verification of fit) a greater reduction in queue time can be expected than that predicted by an inappropriate service model.

Following are the two server versions.





As with the single server case, simulated mean queue time with normally distributed service time is roughly half that of exponential service time (.033 minutes or 1.98 sec, vs. .059 minutes or 3.54 sec). Both distributions also exhibit a right tail, but not as pronounced as in the single server trial (with more servers, we expect a more efficient queue), indicating that a significant reduction in maximum queue time would be expected by adding another server, regardless of the service model.

Here's the program. SQL naturally handles creating, deleting, and updating records very well (which models customers entering and leaving a queue and servers beginning and terminating service), and it supports most common mathematical functions, including $\mathbf{e}^t$ and **log**(*p*), which are all one needs to calculate exponential probabilities and quantiles, but it (typically) does not have a normal cdf or quantile function, so I had to write my own (based on a combination of integrating the MacLaurin series for $e^{-x^2/2}$ and Newton's method for the inverse cdf). I considered a re-write in SAS (since it has plenty of cdf and inverse support), but its SQL implementation is a bit strange (ANSI, but peculiar), plus my while loops and variable passing would have been in macro statements which, although necessary in SAS, are cumbersome and add no value to data, computations, or results. Furthermore, I now have a compact (ten lines, four of which are *begin* or *end*) normal cdf and inverse function in SQL. I highlighted that section, if you are interested.

```
create proc TollBoothSimulator @epochEnd real, @logtrans varchar(5), @logdata varchar(5) as

-- Simulate toll station with multiple servers (booths, pay stations, etc.)
-- Generate service capacities and proportion incoming volume served by various service styles
-- Accumulate total elapsed time, time each server provides server, and time arriving traffic is in
-- queue
-- Note time, here, is not clock time but a quantity of minutes from some epoch

-- @epochEnd is the desired duration of simulation run
-- Arrivals are modeled as Poisson distributed with parameter @lambda (cars per minute)
-- Servers of style 'exp' are modeled with exponential service times using parameter @mu (mu = expected
-- cars serviced per minute)

-- Outline:
-- 1.  Initialize current epoch (to 0 minutes)
-- 2.  Set each server status to available
-- 3.  Randomly generate customer arrival times (from arrival probability distribution) through the
--     specified end epoch
--     Customers are effectively in queue but will not be recognized before the epoch reaches their
--     arrival time
-- 4.  Set current epoch to time of first arrival
-- 5.  While time left in simulation
--       While customers in queue and server available
--         Assign next customer in queue (earliest arrival) to randomly selected available server
--         Generate random expected service completion time (current epoch + duration) from service
--         probability distribution
--         Set service start time to current epoch
--         Flag selected server as unavailable
--         Calculate and accumulate customer's time in queue (current epoch - arrival) for analysis
--       Advance epoch to minimum of next completion time of all servers and earliest arrival in queue
--       While there are busy servers with completion times = current epoch
--         Complete service in progress
--         Flag busy servers as available
--         Accumulate completions and service times by server (service time = completion time - start
--         time)
-- 6. Calculate final statistics:  percent service time (utilization) and queue time (average per
--    customer)

set nocount on

declare @epoch real, @ts real, @ta real, @i smallint, @j smallint, @n smallint, @CustomersServed int,
        @CustomerQtime real, @lambda real, @p real, @paccum real, @dist varchar(10), @par1 real, @par2
real
declare @Q table(id int identity, ArrivalTime real)
declare @Server table(id smallint, Pbound0 real, Pbound1 real, ServiceDist varchar(5), Par1 real,
        Par2 real, Available bit, ServiceStartTime real, ServiceCompletionTime real, ServiceCount int,
        TotalServiceTime real)
select  @CustomersServed=0, @CustomerQtime=0

-- Normal probability distribution approximation resources
declare @z real, @Fe real, @Fterm real, @k smallint, @abovemean bit

if(@logtrans='yes') truncate table tblog
```

```
-- Randomly generate arrivals
-- Note that each new arrival time is based off of the previous one (exponential interarrival times)
-- Get exponential distribution parameter
select @lambda=Parameter1 from TollBoothArrivals where Active=1 and PDistribution='exp'
select @ta=-log(1-rand())/@lambda
while(@ta<@epochEnd)
  begin
    insert into @Q values(@ta)
    select @ta=@ta-log(1-rand())/@lambda
  end

if(@logtrans='yes') insert into tblog select 0, 'addq', id, 0, ArrivalTime, 0 from @q order by id

-- Create servers, assign probability distributions and probability of service from server cfg table
-- Probability of service band is cumulative probabilities of previously configured servers plus p
-- Ensure that cumulative pvalues sum to 1
-- Make available

select @i=0, @paccum=0
declare c cursor for select Quantity, ProportionCustomersServed, PDistribution, Parameter1, Parameter2
from TollBoothServers
open c
fetch next from c into @n, @p, @dist, @par1, @par2
while(@@fetch_status=0)
  begin
    select @j=1
    while(@j<=@n)
      begin
        select @i=@i+1
        insert into @Server(id, PBound0, Pbound1, ServiceDist, Par1, Par2, Available, ServiceStartTime,
        ServiceCompletionTime, ServiceCount, TotalServiceTime)
        values(@i, @paccum, @paccum+@p, @dist, @par1, @par2, 1, 0, 0, 0, 0)
        select @paccum=@paccum+@p
        select @j=@j+1
      end
    fetch next from c into @n, @p, @dist, @par1, @par2
  end
deallocate c

-- Begin simulation
select @epoch=0
while(@epoch<@epochEnd)
  begin

    -- Advance epoch to minimum of earliest service completion time and next arrival beyond current
    -- epoch
    select @ts=min(ServiceCompletionTime) from @Server where Available=0
    select @ta=min(ArrivalTime) from @Q where ArrivalTime>@epoch
    select @epoch=case when(@ts is not null and @ta is not null)then
                         case when(@ts<@ta)then @ts
                              else @ta
                         end
                       else isnull(@ta, @ts)
                  end

    if(@logtrans='yes') insert into tblog select @epoch, 'epochadv', 0, 0, @epoch, 0

    -- Complete all service in progress with completion time <= current epoch
    -- Accumulate service statistics
    if(@logtrans='yes') insert into tblog
                        select @epoch, 'srvcompl', id, 0, 0, 0
                        from   @Server where  Available=0 and ServiceCompletionTime<=@epoch
    update @Server
    set    Available=1, ServiceCount=ServiceCount+1,
           TotalServiceTime=TotalServiceTime+ServiceCompletionTime-ServiceStartTime
    where  Available=0 and ServiceCompletionTime<=@epoch

    -- Assign in-queue customers (arrival time <= current epoch, in arrival order) to randomly selected
    -- servers
    while(exists(select * from @Q where ArrivalTime<=@epoch)
           and exists(select * from @Server where Available=1))
      begin
        -- Randomly select server (the one with random number in its p-band)
        -- Note that the selected server might be occupied, in which case no assignment is made
```

```
select @p=rand(), @i=0
select @i=id, @dist=ServiceDist, @par1=Par1, @par2=Par2
from   @Server
where  Available=1 and @p>=Pbound0 and @p<Pbound1
-- Generate service completion time (from specified probability distribution) if serve available
if(isnull(@i,0)>0)
  begin
    -- Record service start (now) and completion times and make server unavailable
    if(@dist='exp')
      update @Server set ServiceStartTime=@epoch, Available=0,
                         ServiceCompletionTime=@epoch-log(1-rand())/@par1
      where id=@i
    else if(@dist='npd')
      -- Randomly generate a normally distributed service time (between avg+-3.5s to avoid
      -- extremes)
      begin
        -- Adjust to right half of distribution and compress to [.001,.499] to eliminate extreme
        -- values
        -- Retain indicator of whether random p-value above or below that for mean (.5 for
        -- normal cdf)
        select @p=rand()-.5
        while(@p not between -.499 and .499)
          select @p=rand()-.5
        if(@p<0)
          select @p=-@p, @abovemean=0
        else
          select @abovemean=1
        -- Adjust random density by npd constant so that following MacLaurin series is
        -- simply for the integral of e**-(x**2/2)
        -- Use initial quantile estimate of .05 (near where most normally distributed values
        -- should be on unit npd)
        -- Note that initial density estimate (Fe) value (as with all following) has been
        -- adjusted for the npd constant
        select @p=@p*sqrt(8*atan(1)), @z=.05, @Fe=0.04997917
        -- Use Newton's method of roots to locate z corresponding to normal cdf within .0001 of
        -- generated p
        while(abs(@Fe-@p)>.0001)
          begin
            -- Calculate cumulative ensity (F) at current quantile estimate (z)
            select @Fe=@z, @Fterm=@z, @k=0, @j=1
            while(@k=0 or abs(@Fterm)>.0001)
              begin
                -- This may appear cryptic, but it implements the Mac series by multiplying the
                -- current term by
                -- the proper factor to generate the next term as z**(2k+1)/[(2k+1)*(2**k)*k!]
                -- as required by Mac
                -- while improving efficiency and avoiding large numerators and denominators
                --(consider z**51/50!)
                select @Fterm=@Fterm*@z*@z*(@k+@k+1)/(@k+@k+2)/(@k+@k+3), @j=-@j
                select @Fe=@Fe+@j*@Fterm, @k=@k+1
              end
            -- Estimate new z by subtracting from current estimate (difference in
            -- F1 and F0)/(cdf derivative)
            -- (this Newton's method of roots)
            -- Note that the derivative of a cdf is its pdf
            if(abs(@Fe-@p)>.0001) select @z=@z-(@Fe-@p)*exp(@z*@z/2)
        end
        -- Assign service time (mean+-zs, subtract from mean if random value was less than .5)
        -- Convert completion times prior to start time to atrt time
        update @Server set ServiceStartTime=@epoch, Available=0,
               ServiceCompletionTime =
               case when(@z*@par2<@par1 or @abovemean=1)then
                      @epoch + @par1 + @z * @par2 * case when(@abovemean=1)then 1 else -1 end
                    else
                      @epoch
               end
        where  id=@i
      end
    -- Get next customer in queue, accumulate queue time, and remove from queue
    select top 1 @j=id from @Q where ArrivalTime<=@epoch order by ArrivalTime
    select @CustomersServed=@CustomersServed+1, @CustomerQtime=@CustomerQtime+@epoch-ArrivalTime
    from   @Q where id=@j
    if(@logtrans='yes')
      begin
        insert into tblog select @epoch, 'srvassgn', @j, @i, ServiceStartTime,
```

```
                                         ServiceCompletionTime
                from   @Server
                where id=@i
                insert into tblog select @epoch, 'qexit', @j, 0, 0, 0
              end
            delete @Q where id=@j
          end
      end

  end

-- Return results
if(@logdata='yes')
  insert into tbdata values(case when(@CustomersServed>0)then @CustomerQtime/@CustomersServed else 0 end)
else
  begin
    select @epochEnd as TotalTime, @CustomersServed as CustomersServed,
           case when(@CustomersServed>0)then @CustomerQtime/@CustomersServed else 0 end as AvgQtime
    select * from @q
    select * from @server
  end
```