ABSTRACT

Some of the most natural uses of queuing theory are inherent in computer science applications. For instance, on-line inquiry systems, requests for records from DASD, order entry and message buffering systems are all examples of a queuing system. Steady state formulas for various systems are described in [1], Probability, Statistics, and Queuing Theory with Computer Science Applications by A. D. Allen. Straightforward SAS code for $M/M/C$, $M/M/1/K$ and a closed Jackson-type multiserver model will be shown.

INTRODUCTION

Results for standard queuing models have been used by operations researchers for years [2,3], however new interest in these models has been generated by performance analysts for tuning and capacity planning applications. The purpose of this paper is to illustrate how easy it is to generate known steady state results with SAS.

The basic elements of a steady state queuing system are shown in Figures 1 and 2 (from [1]). The principal quantities of interest are: $N$, $N_q$, $N_s$, $t$, $w$, and $a$. All are random variables with probability distributions. Thus to describe the system we will want to determine the steady state distributions and/or moments and percentiles for each of the counts and waiting times.

We will also assume we know or can estimate the number of servers in the service facility and $E(t)$ and $E(s)$. If these are assumed known as inputs we therefore also know:

\[ \lambda = \frac{1}{E(t)} \text{... average arrival rate} \]
\[ \mu = E(s) \text{... average service rate} \]
\[ \rho = \lambda / \mu = \text{traffic intensity} \]
\[ \phi = \mu / c = \text{server utilization} \]

By taking expected values of equations (1) and (2) we get:

\[ EN = EN_q + EN_s \]
\[ EN_q = Wq + Lq \]
\[ EN_s = Wq + Lq \]

And by Little's result if $\lambda$ and any one of $N$, $Wq$, $Lq$ is known then we can determine the other three. Little's result for steady state queues under general conditions is $L = \lambda W$ and $Lq = \lambda Mq$. We define the following terms:

- $N$ = the steady state number of customers in the system
- $N_s$ = the steady state number of customers in the waiting line
- $N_q$ = the steady state number of customers in the service facility
- $\tau$ = the interarrival time
- $w$ = the total time a customer spends waiting in the queue
- $a$ = the service time

Example 1: SAS code for a steady state $M/M/C$ system

Suppose we have an infinite source, exponential interarrival and service times, $C$ servers each with a first come, first serve (FCFS) queue discipline. In the Kendall notation this is denoted as a $M/M/C$ queueing system and is pictured in Figure 2 with $\lambda$ and $\mu$ being exponential interarrival and service rates. Steady state formulas for the $M/M/C$ system appear in Table 5 of Allen's book, [1]. The key quantity to compute is:

\[ P_0 = P(N=0) = \frac{1}{\sum_{n=0}^{c-1} \frac{u^n}{n!} + \frac{\mu}{cl}(1-p)} \]
All other quantities such as Erlang’s c formula, C(c, u)=P\{N] \}, moments, percentiles etc. are easily generated once Pc is computed. To compute Po just compute the successive factorial sums of
\[ C(c+1) \]
\[ C(c+2) \]
\[ \ldots \]
\[ C(c+n) \]
Notice that if we compute in a loop new term = old term \* u/n and new sum = old sum + new term we can obtain the desired sum and hence, Po.

The SAS code is:

MACRO LM \* INPUT PARAMETERS;
MACRO M \* \*;
MACRO C \* \*;
MACRO MMC;

DATA M M C;
SUM=1; TERM=1; MAX=C-1;
U=LM/M; RHO=D/C; FAC=1-RHO;
DO I=1 TO MAX;
TERM=TERM*U/I;
SUM=SUM+TERM;
END;
LTERM=TERM*U/(C*FAC);
P0=1/(SUM+LTERM);
PC=FAC*LTERM*P0;
CCU=1-PC/FAC;
LAMBDA-LM; MU=M; SERVERS=C;
ES=1/MU; FACSQ=FAC*FAC; CSQ=C*C;
THRUPT= RHO*MU; *AVERAGE THRUPT;

--EXPECTED VALUES--
LQ=U*CCU/(C*FAC);
WQ=LQ/LAMBDA;
L =LQ=U;
W =L/LAMBDA; *AVE RESPONSE TIME;

--VARIANCES--;
VARSQ=2-LCCU*CCU*ES*ES/(CSQ*FACSQ);
VARQ=ES*ES*ES*(1-RHO-RHO*CCU)/FACSQ;
IF U=MAX
THEN ESQ=ES*(1-ESQ*ES/CCU) ; ELSE DO;
    NUM=2*ES*ES*(ESQ*(1-ESQ)*FACSQ));
    DEN=(U+1)*ESQ*FACSQ);1;
    ESQ=NUM/DEN; END;
    VARSQ=ESQ*ESQ;

--PERCENTILES FOR S. S. Q DISTRIB--;
P = ES*(CCU); Q5=ES*LN(2*CCU); Q7=ES*LN(4*CCU); Q9=ES*LN(16*CCU); Q95=ES*LN(100*CCU);

--PRINT RESULTS--;
PROC PRINT;
VAR LAMBDA MU SERVERS RHO U CCU THRUPT;
L Q W WQ;
VARNO VARG VARQ Q5Q-99;
TITLE M/M/C--STEADY STATE;
TITLE2 PARAMETERS, EXPECTED VALS & VARS;
TITLE3 AND SELECTED PERCENTILES FOR Q; Z

Example 2: A capacity planning question, “How many servers do I need?”

Consider the same M/M/C system. To answer a question like “How many servers do I need such
that the probability that wait time (response time) is less than 3 seconds 90% or more of the time?” we need to know W(t)=P\{w=t\} as a function of c. Unfortunately we can not explicitly solve for c since:

\[ \frac{\text{LH}_c e^{-\mu t} e^{\text{CCU}(t-\mu)} \mu e^{-\mu t}}{1-[\text{LH}_c e^{\text{CCU}(t-\mu)} \mu e^{-\mu t}]} \text{ if } \mu c<1 \]

\[ \frac{1}{1-[\text{LH}_c e^{\text{CCU}(t-\mu)} \mu e^{-\mu t}]} \text{ if } \mu c=1 \]

C1 and C2 are both functions of C and (c,u). We can however increment C while W(t)>.90 with the DO WHILE statement of SAS79.3. The code below assumes we are dealing with data set M M C generated in Example 1. It also assumes that u/c=1.

MACRO TG t X;
MACRO PCT p X;

DATA; SET;
PW=LT TG=0;
DO WHILE (PW_LT TG<PCT);
C=C+1;
SUM=SUM+TERM+FAC;
LTERM=TERM*U/C; *AVERAGE LTERM;
PC=FAC*LTERM*P0;
CCU=1-PC/FAC;
LAMBDA-LM; MU=M; SERVERS=C;
ES=1/MU; FACSQ=FAC*FAC; CSQ=C*C;
THRUPT= RHO*MU; *AVERAGE THRUPT;

--EXPECTED VALUES--
LQ=U*CCU/(C*FAC);
WQ=LQ/LAMBDA;
L =LQ=U;
W =L/LAMBDA; *AVE RESPONSE TIME;

--VARIANCES--;
VARSQ=2-LCCU*CCU*ES*ES/(CSQ*FACSQ);
VARQ=ES*ES*ES*(1-RHO-RHO*CCU)/FACSQ;
IF U=MAX
THEN ESQ=ES*(1-ESQ*ES/CCU) ; ELSE DO;
    NUM=2*ES*ES*(ESQ*(1-ESQ)*FACSQ));
    DEN=(U+1)*ESQ*FACSQ);1;
    ESQ=NUM/DEN; END;
    VARSQ=ESQ*ESQ;

--PERCENTILES FOR S. S. Q DISTRIB--;
P = ES*(CCU); Q5=ES*LN(2*CCU); Q7=ES*LN(4*CCU); Q9=ES*LN(16*CCU); Q95=ES*LN(100*CCU);

--PRINT RESULTS--;
PROC PRINT;
VAR TG PW C;
TITLE C=#SERVERS FOR;
TITLE2 P(W<=TG»==PCT;

Example 3: Finite population model of an interactive computer system
(M/M/1/K steady state queue)

Figure 3 illustrates our next model. The assumptions are that each customer is in one of 3 states:

o thinking at the terminal
o queuing for CPU service
o receiving CPU service

If think and service times are exponential this is the classical machine repair model, M/N/1/K, with repairman corresponding to CPU server and machine breakdown corresponding to service from terminals.

Figure 3.
For this example for purposes of illustration we will only calculate:

\[ P_0 = P(N=0) \]

where

\[ P_0 = 1 - \text{mean response time subtracting off } \gamma_0 \text{ think time} \]

\[ W = K \text{ mean response time subtracting off } \gamma_0 \text{ think time} \]

\[ AT = \frac{1}{\lambda} + \frac{1}{\mu_1} \text{ average throughput} \]

\[ N_{\text{sat}} = 1 + \frac{\mu_1}{\lambda} \text{ system saturation} \]

Scherr [5] had excellent results in applying this model to analyze an early time sharing system, CTSS, at MIT. This was a system in which user programs were swapped in and out of memory with only one complete program in memory at a time. Since there was no overlap of program execution and swapping, Scherr used the sum of program execution time and swapping time as the CPU service time. The machine repair analytic model gave results that were very close to those for a simulation model and to actual results.

Once again, similar to example 1, \( P_0 \) generates everything where:

\[ P_0 = \left( \frac{1}{\lambda} \sum_{i=0}^{K-1} \frac{\lambda^{i+1}}{i!} \right)^{-1} \]

And once again we need only expand the partial sums 1, \( 1 + \frac{1}{\mu_1} \), \( 1 + \frac{1}{\mu_1} + \frac{1}{\mu_2} \), ..., to see how to generate the factorial sums successively. The SAS code is:

MACRO LN \lambda \% 
MACRO KM \mu \% 
MACRO N \% 
MACRO MM \% 
DATA MACHREP: 
SUM=1; TERM=1; LAMBDA=K; 
MU=\%M; K=\%K; 
DO J=1 TO K; 
TERM=(K+1-J)*X*TERM; 
SUM=SUM+TERM; 
END; 
PO=1/SUM; 
RHO=1-PO; 
W=(K/K+1)-1/LAMBDA; 
AVETHRPT=RHO*MU; 
N_AVE=AVETHRPT+X1; 
N_SAT=1+MU/LAMBDA; 
PROC PRINT: VAR LAMBDA MU RHO W AVETHRPT N_AVE N_SAT ;

Example 4: Central Server Model of Multiprogramming

This model is in Figures 4 and 5. It is a closed model since it contains a fixed number of programs which can be thought of as markers that cycle around the system interminably. However, each time a marker (program) makes the cycle from CPU directly back to the CPU we assume a program execution has been completed and a new program entered the system. If we consider Figure 5 to be the central processor system of Figure 4 then we must assume the users at the terminals are sufficiently active to guarantee there is always an interaction pending. Thus we could also classify the central server model as an infinite source model.

We further assume there are \( M-1 \) I/O devices each with exponential service rate \( \mu_i \) (\( i=2,3, \ldots M \)) and the CPU also provides exponential service with rate \( \mu_1 \). If \( K=(K_1, K_2, \ldots, K_M) \) is the state of the system, \( K_i \) is the number of jobs at each queue, then Buzen [6] shows that:

\[ P(K=K_1, K_2, \ldots, K_M) = \frac{1}{K_M} \prod_{i=2}^{M} \frac{K_i}{\mu_i} \]

In this example we are interested in server utilizations, average throughput and average response time where:

\[ G(K-1)/G(K) \text{ server utilizations} \]

\[ \gamma_i = 1/\mu_i \text{ average throughput} \]

\[ W = \lambda \text{ average response time} \]

And where:

\[ G(K) = g(K, M) \]

\[ g(K, M) = g(K-1, M) + x_j, g(M-1, M) \]

\[ x_j = \gamma_j/\mu_j \]

and \( G(K) \) is computed by Buzen’s algorithm [6] found in appendix 1. \( G(K) \) in fact is the key quantity to compute and this is easily accomplished using PROC MATRIX.
Questions:  How do I know interarrival and/or service times are exponential?  On a primary level one could just work with the rates $\lambda$ and $\mu$ and assume they were exponential rates.  The steady state results would not necessarily be correct, although many steady state results are fairly robust.  The better approach is to statistically test to see if a process is Poisson (i.e. look at $\lambda$ and $\mu$ and see what effects there are on average throughput, the 95th percentile of response time, etc.) and capacity planning (i.e. how many servers do I need to get desired response time or throughput)?

Conclusion

We have seen that it is easy to generate some simple steady state queuing theory results with SAS and that this is a natural by-product of the normal reduction process an analyst is already performing.  I hope these examples will encourage others to generate SAS code for other queuing system models.

Acknowledgements

I would like to express appreciation to Dr. Arnold O. Allen for writing such a lucid text [1] on this subject and to Mr. Ron Teutsch and Mr. William Mullen for pointing out this book to me.  I also thank Academic Press and Dr. Allen for allowing me to reproduce the figures and Buzen's Algorithm from the book.

Question:  Why use queuing theory models at all?  A model is only as good as its predictions.  For many real-life situations an analytic model is too simplifying.  Other approaches including simulation may prove fruitful in these cases.  However, if a queuing theory model does prove useful it will allow the analyst to do sensitivity analysis (i.e. vary $\lambda$ and $\mu$ and see what effects there are on average throughput, the 95th percentile of response time, etc.) and capacity planning (i.e. how many servers do I need to get desired response time or throughput?).

PROC PRINT DATA=UTILIZATION;
   TITLE SERVER UTILIZATION;
PROC PRINT DATA=RESPONSE;
   TITLE THROUGHPUT & MEAN RESPONSE TIME;
Final Comments

Question:  How do I know interarrival and/or service times are exponential?  On a primary level one could just work with the rates $\lambda$ and $\mu$ and assume they were exponential rates.  The steady state results would not necessarily be correct, although many steady state results are fairly robust.  The better approach is to statistically test to see if a process is Poisson with a Lilliefora-Stephens test [7].  See appendix 2 for a SAS macro LSEXP which performs this vital test.  Allen [1] suggests looking at $C V^2 = \text{Var}X / \left(\text{EX}^2\right)$ for an indication of whether or not a process is Poisson (or equivalently whether the successive wait times are exponential) where the indication and value of $CV^2$ are as follows:

- $0...\text{constant}$
- $1...\text{exponential}$
- $1/k...\text{Erlang}\times\Gamma(k,1)$
- $1...\text{hypergeometric}$

Last, choose the appropriate A/B/K/M/2 system and use the appropriate steady state formulas to generate results when interarrival and/or service times are not exponential.

Question:  Why use SAS for these probability models?  Perhaps one of the most basic arguments for a performance analyst is that if you are going to reduce the data anyway (with MEANS, UNIVARIATE, or SUMMARY) then it is very easy to go one step further with code like examples 1-3 to get desired average throughput, average response time, number of servers required, etc.  Unlike the examples where the rates were passed in through macros, the user can easily output means, take reciprocals to get rates and pass these rates into the dataset that generates the results.  For instance, in examples we could generate rates $\lambda$ and $\mu$ as reciprocals from PROC MEANS in a previous step.  Then instead of having to define the values $\lambda$ and $\mu$ outside as macros we can easily pass them into the DATA step as:

MACRO C = $X$;
DATA M_M_C;
   RETAIN LAMBDA MU;
   IF N = 1 THEN SET;
   MACRO C = $X$;

PROC PRINT DATA=UTILIZATION;
   TITLE SERVER UTILIZATION;
PROC PRINT DATA=RESPONSE;
   TITLE THROUGHPUT & MEAN RESPONSE TIME;

Final Comments

Question:  How do I know interarrival and/or service times are exponential?  On a primary level one could just work with the rates $\lambda$ and $\mu$ and assume they were exponential rates.  The steady state results would not necessarily be correct, although many steady state results are fairly robust.  The better approach is to statistically test to see if a process is Poisson with a Lilliefora-Stephens test [7].  See appendix 2 for a SAS macro LSEXP which performs this vital test.  Allen [1] suggests looking at $C V^2 = \text{Var}X / \left(\text{EX}^2\right)$ for an indication of whether or not a process is Poisson (or equivalently whether the successive wait times are exponential) where the indication and value of $CV^2$ are as follows:

- $0...\text{constant}$
- $1...\text{exponential}$
- $1/k...\text{Erlang}\times\Gamma(k,1)$
- $1...\text{hypergeometric}$

Last, choose the appropriate A/B/K/M/2 system and use the appropriate steady state formulas to generate results when interarrival and/or service times are not exponential.

Question:  Why use SAS for these probability models?  Perhaps one of the most basic arguments for a performance analyst is that if you are going to reduce the data anyway (with MEANS, UNIVARIATE, or SUMMARY) then it is very easy to go one step further with code like examples 1-3 to get desired average throughput, average response time, number of servers required, etc.  Unlike the examples where the rates were passed in through macros, the user can easily output means, take reciprocals to get rates and pass these rates into the dataset that generates the results.  For instance, in example
Algorithm 6.3.1 (Buzen's Algorithm)

Given the parameters of the central server model of Fig. 6.3.4 (that is, $\mu_i$, $p_i$ for $i = 1, 2, \ldots, M$), the algorithm will generate $G(K) \equiv G(0, M)$ defined by (6.2.23) as well as $g(a) \equiv G(a - 1)$.

\[ G(K - 2), \ldots, G(1), G(0) = 1. \]

Step 1 [Assign values to the $x_i$] Set $x_1 = 1$ and then set $x_i = \mu_i p_i$, $i = 2, 3, \ldots, M$.

Step 2 [Set initial values] Set $g(0, 1) = 1$ for $k = 0, 1, \ldots, K$ and set $g(0, m) = 1$ for $m = 1, 2, \ldots, M$.

Step 3 [Initialize k] Set $k$ to 1.

Step 4 [Calculate new row] Set $g(k, m) = g(k, m - 1) + x_m g(k - 1, m)$, $m = 2, 3, \ldots, M$.

Step 5 [Increase k] Set $k$ to $k + 1$.

Step 6 [Algorithm complete?] If $k \geq K$ return to Step 4. Otherwise terminate the algorithm. Then $g(n, M) = G(n)$ for $n = 0, 1, \ldots, K$.

References


