

ANALYTIC PERFORMANCE MODELS FOR SINGLE CLASS AND MULTIPLE CLASS MULTITHREADED SOFTWARE SERVERS

DANIEL A. MENASCÉ*
DEPT. OF COMPUTER SCIENCE
GEORGE MASON UNIVERSITY
FAIRFAX, VA 22030, USA
MENASCE@CS.GMU.EDU

MOHAMED N. BENNANI
ORACLE, INC.
1211 SW FIFTH AVENUE
PORTLAND, OR 97204
MOHAMED.BENNANI@ORACLE.COM

Abstract

Modern computer systems are based on a wide variety of software servers, such as web servers, application servers, database servers, and mail servers. The typical software architecture of such servers includes a set of processes or threads that serve requests submitted to the server. Requests that arrive at the server and find all threads busy, are placed in a queue. Threads that are busy executing requests compete for hardware resources (e.g., processors and I/O devices) at the machine where the software server runs. It is important to be able to model software servers in a way that takes into account both software contention—waiting for threads—and hardware contention—waiting for processors and I/O devices. This paper presents analytical models for a wide range of multithreaded software server architectures: a) single class (i.e., all requests have similar demands) with unlimited thread queue size, b) single class with limited thread queue size, c) multiclass with unlimited thread queue size, and d) multiclass with limited queue size. Numerical results are presented to illustrate the use of the models.

1 Introduction

Modern computer systems are based on a wide variety of software servers that serve requests from client processes. Some examples of software servers include web servers, application servers, database servers, and mail servers. The typical software architecture of such servers includes a set of processes or threads that serve requests submitted to the server. Requests that arrive at the server and find all threads busy, are placed in a queue. Threads that are busy executing requests compete for hardware resources (e.g., processors and I/O devices) at the machine where the software server runs.

Therefore, it is important to be able to model software servers in a way that takes into account both software contention—waiting for threads—and hardware contention—waiting for processors and I/O devices. This paper presents analytical models for a wide range of multithreaded software server architectures: a) single class (i.e., all requests have similar demands) with unlimited thread queue size, b) single class with limited thread queue size, c) multiclass with unlimited thread queue size, and d) multiclass with limited queue size.

The models presented here build on well-known modeling techniques: birth-death Markov chains [1], Mean Value Analysis (MVA) [4] for single class queuing network models, and approximate MVA (aMVA) [5] for multiple class queuing network models. We also use the well-known principles of hierarchical modeling and iterative models.

The rest of this paper is organized as follows. Section two presents some background and notation. Section three presents a model for a single class software server with an unlimited thread queue size. The next section modifies the model to account for thread queues of limited size. Section five deals with the issue of multiple classes of requests for both unlimited and limited queue size. Finally section six presents some concluding remarks.

2 Basic Concepts and Notation

The models used in this paper draw on the notion of closed queuing network (QN) models used to represent computer systems. Such models comprise a collection of K devices visited by requests that may belong to different classes, or workloads. A class of requests is typically associated to a collection of requests that have similar demands on the various resources. The service demand, $D_{i,r}$, of requests of class r at device i is the total service time spent by a request of class r over all visits to device i during the

*This work was supported in part by the National Geospatial-Intelligence Agency (NGA) contract NMA501-03-1-2033.

execution of the request. The solution to such QNs can be obtained through Mean Value Analysis [4] or approximate MVA (aMVA) for the case of multiple classes [5]. For more details on queuing networks, the reader is referred to [3]. The notation given in Table 1 is used throughout the paper.

3 Single-class Multithreaded Server with Unlimited Queue

Figure 1 displays a situation in which requests for service arrive to a multithreaded software server that has M threads. The underlying hardware platform used by all threads has processing and I/O resources. Arriving requests join a queue of requests waiting for an available thread. Once a thread begins to execute, it starts to compete with other threads for processing and I/O resources. The bottom part of the figure shows queues for the CPU and for I/O. It is also assumed in this section that all requests are homogeneous in terms of their service demands. This assumption will be relaxed later in the paper. Thus, we are dealing with a single-class modeling situation.

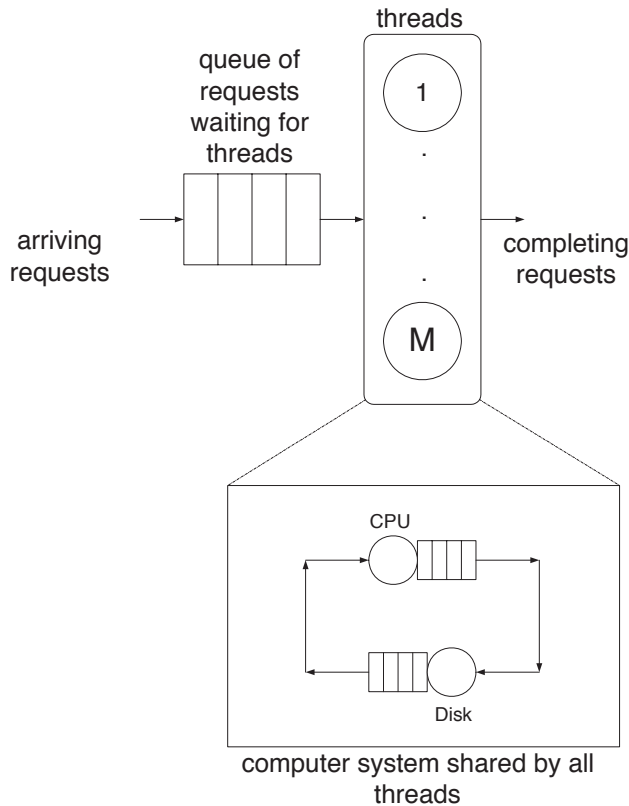


Figure 1: Multithreaded server with unlimited queue.

In order to model this type of software server, we use the well-known principle of hierarchical modeling. At the software level, we use a Markov chain to model the arrival

and completion process of requests. The rate at which requests complete is determined by the solution of a lower level model, in the form of a queuing network (QN).

Let $X(k)$, $k = 1, \dots, M$, be the rate at which a thread completes its execution when there are k requests in execution. This rate can be obtained by solving the closed QN model for the hardware subsystem composed of the CPU and disk when there are k concurrent threads in execution. The solution of this queuing network can be obtained through Mean Value Analysis (MVA) [3, 4].

A state k in the Markov chain of Fig. 2 represents the number of requests in the system (waiting for a thread or being executed by a thread). This Markov chain has an infinite number of states since there is no limit on the number of requests in the queue. The arrival rate in the Markov chain (transition rate to a higher numbered neighboring state) is the arrival rate of requests, λ , and the completion rate (transition rate to a lower numbered neighboring state) is the completion rate of threads. Note that since there can be at most M threads in execution, the departure rate is constant and equal to $X(M)$ for any state $k > M$.

The solution of this Markov chain, i.e., the set of probabilities P_k , $k = 0, \dots$, can be obtained using the Generalized Birth-Death theorem [1, 3]:

$$P_k = P_0 \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}}. \quad (1)$$

For the model considered in this subsection, $\lambda_i = \lambda \forall i$ and $\mu_i = X(i)$ for $i = 0, \dots, M$ and $\mu_i = X(M)$ for $i > M$.

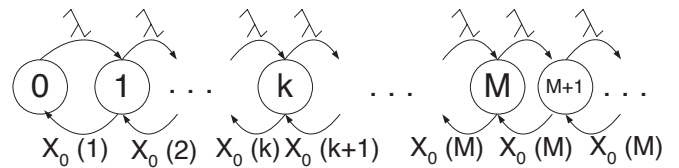


Figure 2: Markov Chain for the unlimited thread queue case.

The solution to the Markov chain of Fig. 2 is given by

$$P_k = \begin{cases} P_0 \lambda^k / \beta(k) & \text{for } k = 1, \dots, M \\ P_0 \rho^k X(M)^M / \beta(M) & \text{for } k > M \end{cases} \quad (2)$$

where:

- $\beta(k) = X(1) \times X(2) \times \dots \times X(k)$,
- $\rho = \lambda / X(M)$, and
- $P_0 = [1 + \sum_{k=1}^M \frac{\lambda^k}{\beta(k)} + \frac{X(M)^M}{\beta(M)} \times \frac{\rho^{M+1}}{1-\rho}]^{-1}$

Note that the solution given above only exists when $\rho < 1$. The average response time, R_0 , can be easily computed from the state probabilities and from Little's Law [1] as

K	Number of devices (e.g., processors, disks) in the computer system
R	Number of classes (workloads).
M	total number of threads in the system
N	system size (maximum queue length plus number of threads in system).
$D_{i,r}$	service demand of class r requests at device i .
$U_{i,r}$	utilization of device i by class r requests.
U_i	utilization of device i ($U_i = \sum_{r=1}^R U_{i,r}$).
$\bar{n}_{i,r}$	average number of class r requests queued at device i or using the device.
\bar{n}_r^{exec}	average number of class r requests in execution ($\bar{n}_r^{\text{exec}} = \sum_{i=1}^K \bar{n}_{i,r}$).
\bar{n}_r^{systr}	average number of class r requests in the system (waiting for a thread or using a thread)

Table 1: Notation.

$R_0 = \bar{n}^{\text{systr}}/\lambda$, where \bar{n}^{systr} , the average number of requests in the system (waiting for a thread or using a thread), is given by

$$\bar{n}^{\text{systr}} = P_0 \left[\sum_{k=1}^M k \frac{\lambda^k}{\beta(k)} + \frac{\rho (X(M) \cdot \rho)^M (1 + M(1 - \rho))}{\beta(M) (1 - \rho)^2} \right] \quad (3)$$

The average number of requests in execution, \bar{n}^{exec} , is given by

$$\bar{n}^{\text{exec}} = \sum_{k=1}^{M-1} k \times P_k + M \times [1 - \sum_{k=0}^{M-1} P_k]. \quad (4)$$

Figure 3 shows the variation of the average response time R_0 as a function of the average arrival rate λ . The parameters used for this example are:

- Number of threads (M): 20
- Service demand at the CPU (D_{CPU}): 0.020 sec
- Service demand at the disk ($D_{\text{I/O}}$): 0.015 sec

For the parameters given above, $X(20) = 49.96$ requests/sec, close to the maximum bound on throughput which is $1/D_{\text{CPU}} = 1/0.02 = 50$ requests/sec. Since $\rho = \lambda/X(20) < 1$, $\lambda < 49.96$. As the figure shows, as λ approaches its maximum possible value, the response time goes to infinity very fast. The response time stays below 0.5 sec for $\lambda < 49$ requests/sec.

4 Single-class Multithreaded Server with Limited Queue

Figure 4 displays a multithreaded server with a limited queue for storing incoming requests. The number of threads is M as in the unlimited queue size case but there is a limit N on the total number of requests in the system (waiting for a thread or being executed by a thread).

The solution for this case can be obtained following the same approach as in section 3. The Markov chain model in this case, shown in Fig. 5, has $N + 1$ states. A state k , ($k = 0, \dots, N$), represents the number of requests in the

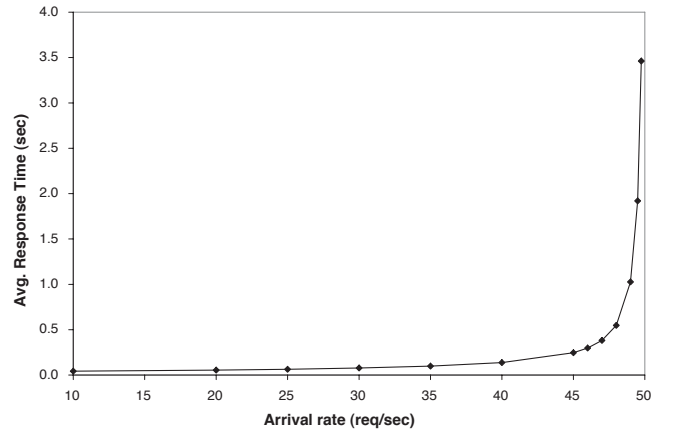


Figure 3: Average response time (sec) vs. arrival rate (req/sec) for the unlimited queue case with 20 threads.

system (waiting for a thread or using a thread). The arrival rate in the diagram is the arrival rate of requests, λ , and the completion rate is the completion rate of threads. Note that since there can be at most M threads in execution, the departure rate is constant and equal to $X(M)$ for any state $k > M$ as in the previous case.

The solution of this Markov Chain, i.e., the values of the probabilities P_k , for $k = (0, \dots, N)$, of finding k requests in the system, can be obtained using the methods in [1, 3]. The solution is given below.

$$P_k = \begin{cases} P_0 \lambda^k / \beta(k) & \text{for } k = 1, \dots, M \\ P_0 \rho^k X(M)^M / \beta(M) & \text{for } k = M + 1, \dots, N \end{cases} \quad (5)$$

where:

- $\beta(k) = X(1) \times X(2) \times \dots \times X(k)$,
- $\rho = \lambda/X(M)$, and
- $P_0 = [1 + \sum_{k=1}^M \frac{\lambda^k}{\beta(k)} + \frac{\rho \times \lambda^M \times (1 - \rho^{N-M})}{\beta(M) \times (1 - \rho)}]^{-1}$

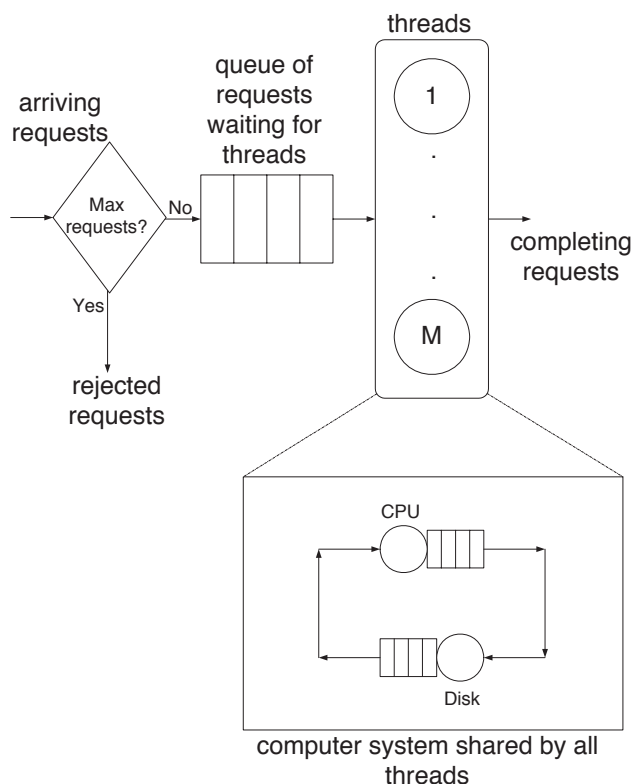


Figure 4: Multithreaded server with limited thread queue.

The metrics of interest, namely the probability of rejection P_{rej} , the average throughput X and the average response time R , can be easily computed from the state probabilities and from Little's Law [1] as follows.

- $P_{rej} = P_n$
- $X = \lambda \times (1 - P_{rej}) = \sum_{k=1}^N X(k) \times P_k$
- $R_0 = \sum_{k=1}^N k \times P_k / X$

Figure 6 shows the variation of the average response time as a function of the average arrival rate for the limited queue case. The parameters for this example are the same

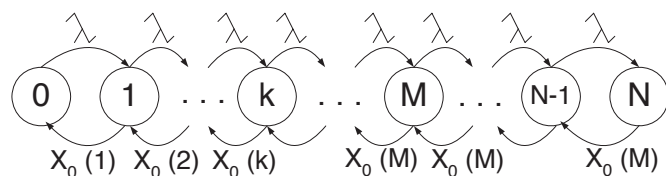


Figure 5: Markov Chain for the limited thread queue case.

used in the unlimited queue case. There is one additional parameter in this case, which is the system size N . The figure shows two curves, one for $N = 25$ and another for $N = 35$.

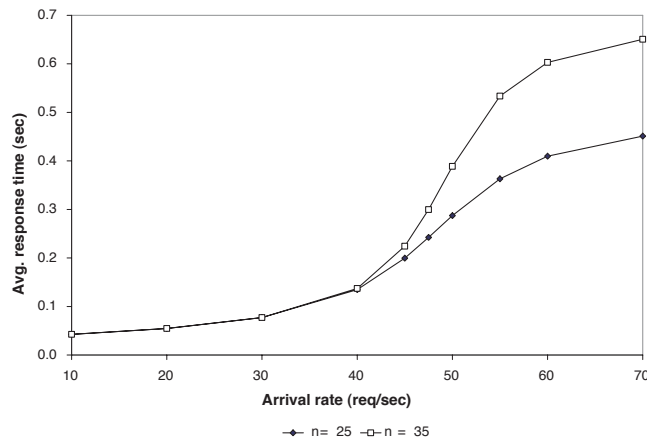


Figure 6: Average response time (sec) vs. arrival rate (req/sec) for the limited queue case with 20 threads and two values of the maximum number of requests in the system ($N = 25$ and $N = 35$).

It is interesting to observe the shape of this curve. At low loads, the probability that requests are lost, P_{rej} , is very low. For example, for $N = 25$ and $\lambda = 45$ requests/sec, $P_{rej} = 1\%$. However, as the load increases, more requests are rejected and the effective arrival rate to the system (equal to the system throughput) increases at a slower rate with λ . For example, for $N = 25$ and $\lambda = 70$ requests/sec, $P_{rej} = 28.7\%$. Thus, only 49.9 requests/sec are actually being accepted by the system for execution. This explains why the response time does not grow to infinity. In fact, for a very large value of λ , accepted requests will find $(N - M) - 1$ requests in the queue for threads and all threads will be busy serving requests. So, the average response time will become constant and equal to the sum of a constant average waiting time and a constant average execution time since all M threads are busy and competing for the hardware resources.

5 Multi-class Multithreaded Server

We now relax the assumption that all requests have the same service demands. Consider R different classes of requests. Class r ($r = 1, \dots, R$) has an arrival rate equal to λ_r requests/sec. There are K devices (e.g., processors and I/O devices) and the service demand of a request of class r at resource i is given by $D_{i,r}$. If we were to use the same two-level modeling approach described in Sections 3 and 4 we would have a multidimensional Markov chain in which

the state would have to indicate the number of requests of each class in execution as well as the number of requests of each class in the queue. This Markov chain would have a very complex and large state space.

Instead of considering such a large state space, we use an approximation that was used by Lazowska et al. [2] to model shared domains in multiprogramming systems. The basic approach works basically as follows:

1. **Initialization:** Estimate the average number of requests in execution, \bar{n}_r^{exec} , for class r and the average number of class r requests in the system (executing or waiting for a thread), \bar{n}_r^{synt} , for all classes $r = 1, \dots, R$.
2. For each class r compute the average number, δ_r^{exec} , of requests of all other classes $r' \neq r$ that are in execution and the average number, δ_r^{synt} , of requests of all other classes $r' \neq r$ that are in the system (in execution or waiting for a thread).
3. Solve a multiclass QN model using approximate Mean Value Analysis (aMVA) [3] where the population vector is $(\bar{n}_1^{\text{exec}}, \bar{n}_2^{\text{exec}}, \dots, n_r, \dots, \bar{n}_R^{\text{exec}})$ for $n_r = 1$ to $M - \delta_r^{\text{exec}}$. In other words, since there are δ_r^{exec} requests in execution from classes other than class r , the number of threads allocated to class r is the difference between the number of threads M and the number of threads allocated to the other classes. The solution of this QN model gives the class r throughput $X_r(\bar{n}_1^{\text{exec}}, \bar{n}_2^{\text{exec}}, \dots, n_r, \dots, \bar{n}_R^{\text{exec}})$ for $n_r = 1$ to $M - \delta_r^{\text{exec}}$.
4. A unidimensional Markov chain such as the ones discussed in sections 3 or 4 can now be built for each class r using λ_r as the arrival rate and the throughputs $X_r(\bar{n}_1^{\text{exec}}, \bar{n}_2^{\text{exec}}, \dots, n_r, \dots, \bar{n}_R^{\text{exec}})$ as the completion rate. The solution of this Markov chain provides new values of the average number of requests in execution \bar{n}_r^{exec} .
5. Repeat steps 2 through 4 until successive values of \bar{n}_r^{exec} are sufficiently close for all classes r .
6. Compute the performance metrics.

We now provide a detailed description of the algorithm to model multiclass multithreaded servers. The pseudocode for the initialization phase (step 1 above) is given in Fig. 7. The purpose of this phase is to obtain an estimate for \bar{n}_r^{exec} and \bar{n}_r^{synt} for all classes. This is done by first computing the utilization of all devices using the Service Demand Law [3]. If all utilizations are below 100%, the formula for multiclass open QNs is used to estimate \bar{n}_r^{exec} . Otherwise, we estimate \bar{n}_r^{exec} using an apportionment factor based on the relative arrival rate of class r requests. In

both cases, we assume that the queue for threads is initially empty, i.e., $\bar{n}_r^{\text{synt}} = \bar{n}_r^{\text{exec}}$. Note that the initial values of \bar{n}_r^{exec} and \bar{n}_r^{synt} will be modified through the iteration process in the remaining steps of the algorithm.

Figs. 8 and 9 provide a detailed description of the iteration phase of the algorithm for the unlimited and limited thread queue cases, respectively. Note that the computation of the probability distributions for P_k follows in each case the formulas used in Sections 3 and 4

```

for every device  $i$  do
   $U_i(\vec{\lambda}) \leftarrow \sum_{r=1}^R U_{i,r}(\vec{\lambda}) = \sum_{r=1}^R \lambda_r * D_{i,r}$ ;
if ( $U_i(\vec{\lambda}) < 1.0$ ) for all  $i$ 
then for every class  $r$  do
  begin
    /* use open QN formula to estimate  $\bar{n}_r^{\text{exec}}$  */
    for every device  $i$  do  $\bar{n}_{i,r} \leftarrow U_{i,r} / (1 - U_i)$ ;
     $\bar{n}_r^{\text{exec}} \leftarrow \sum_{i=1}^K \bar{n}_{i,r}$ ;
    /* adjust  $\bar{n}_r$  so that it does not exceed
        $M_r$ , the number of class  $r$  threads
       estimated to be in execution */
     $\bar{n}_r^{\text{exec}} \leftarrow \min\{\bar{n}_r^{\text{exec}}, M_r\}$ 
    where  $M_r = M \times \bar{n}_r^{\text{exec}} / \sum_{s=1}^R \bar{n}_s^{\text{exec}}$ ;
    /* assume no queuing for threads at
       initialization */
     $\bar{n}_r^{\text{synt}} \leftarrow \bar{n}_r^{\text{exec}}$ 
  end
else begin
  /* alternative initialization: use arrival rate
     ratios to estimate  $\bar{n}_r$  */
  for every class  $r$  do
    begin
       $\bar{n}_r^{\text{exec}} \leftarrow M \times \lambda_r / \sum_{s=1}^R \lambda_s$ ;
      /* assume no queuing for threads at
         initialization */
       $\bar{n}_r^{\text{synt}} \leftarrow \bar{n}_r^{\text{exec}}$ 
    end
  end
end

```

Figure 7: Initialization Phase of the Performance Model for a Multiclass Multithreaded Server.

Table 2 shows the results after three iterations for a two-class multithreaded case with unlimited queue. The following parameters were used: $M = 10$ threads, $\lambda_1 = 25$ requests/sec, $\lambda_2 = 20$ requests/sec, $D_{\text{CPU},1} = 0.014$ sec, $D_{\text{I/O},1} = 0.020$ sec, $D_{\text{CPU},2} = 0.018$ sec, and $D_{\text{I/O},2} = 0.012$ sec. Iteration 0 shows the results of the initialization step. Columns 7 and 13 show the absolute percent relative error between two consecutive values of \bar{n}_r^{exec} . Note that in iteration 3, this error is already at 0.3%. More iterations would yield even smaller errors. The response time for classes 1 and 2 at iteration 3 is 0.1301 seconds and

Iteration step:for every class $r = 1, 2, \dots, R$ do

begin

$$\delta_r^{\text{exec}} = \sum_{r' \neq r} \bar{n}_{r'}^{\text{exec}}$$

/* adjust the value of M */

$$M^* = M - \delta_r^{\text{exec}}$$

for $n_r = 1, 2, \dots, \lfloor M^* \rfloor$ doCall aMVA to compute $X_r(\bar{n}_1^{\text{exec}}, \bar{n}_2^{\text{exec}}, \dots, n_r, \dots, \bar{n}_R^{\text{exec}})$ if M^* is not an integer thenCall aMVA to compute $X_r(\bar{n}_1^{\text{exec}}, \bar{n}_2^{\text{exec}}, \dots, M^*, \dots, \bar{n}_R^{\text{exec}})$ define a single class generalized birth-death model with $\lambda = \lambda_r$ and death rates:

$$\mu_r(k) = X_r(\bar{n}_1^{\text{exec}}, \bar{n}_2^{\text{exec}}, \dots, k, \dots, \bar{n}_R^{\text{exec}}), \text{ for } 1 \leq k \leq \lfloor M^* \rfloor, \text{ and}$$

$$\mu_r(k) = X_r(\bar{n}_1^{\text{exec}}, \bar{n}_2^{\text{exec}}, \dots, M^*, \dots, \bar{n}_R^{\text{exec}}) \text{ for } k > \lfloor M^* \rfloor.$$

compute the probability distribution, P_k , according to Section 3

$$P_k = P_0 \times \frac{\lambda_r^k}{\beta(k)}, \text{ for } 1 \leq k \leq \lfloor M^* \rfloor, \text{ and}$$

$$P_k = P_0 \times \rho^k \times \mu_r(M^*)^{\lfloor M^* \rfloor} / \beta(\lfloor M^* \rfloor), \text{ for } k > \lfloor M^* \rfloor$$

where $\rho = \lambda_r / \mu_r(M^*)$, $\beta(k) = \mu_r(1) \times \dots \times \mu_r(k)$, and

$$P_0 = \left[1 + \sum_{k=1}^{\lfloor M^* \rfloor} \lambda_r^k / \beta(k) + \frac{\mu_r(\lfloor M^* \rfloor)^{\lfloor M^* \rfloor} \times \rho^{\lfloor M^* \rfloor + 1}}{\beta(\lfloor M^* \rfloor) \times (1-\rho)} \right]^{-1}$$

compute the performance metrics as:

average number of requests in the system:

$$\bar{n}_r^{\text{sys}} = P_0 \left[\sum_{k=1}^{\lfloor M^* \rfloor} k \lambda_r^k / \beta(k) + \frac{\rho [\mu_r(\lfloor M^* \rfloor)]^{\lfloor M^* \rfloor} (1 + (M^*) (1-\rho))}{\beta(\lfloor M^* \rfloor) (1-\rho)^2} \right].$$

average throughput:

$$\bar{X}_r = \lambda_r$$

average response time:

$$R_{0,r} = \bar{n}_r^{\text{sys}} / \lambda_r$$

average number of requests in execution:

$$\bar{n}_r^{\text{exec}} = \sum_{k=1}^{\lfloor M^* \rfloor - 1} k \times P_k + M^* \times [1 - \sum_{k=0}^{\lfloor M^* \rfloor - 1} P_k]$$

end

Convergence test step: Repeat the iteration step until successive values of \bar{n}_r^{exec} are sufficiently close for all classes**Compute final metrics** Return the performance metrics as computed in the last iteration

Figure 8: Iterative Phase of the Performance Model for a Multiclass Multithreaded Server with Unlimited Queue.

0.1110 seconds, respectively.

6 Concluding Remarks

This paper showed how well-known analytic performance models can be used to assess the performance of multi-threaded software servers. The models use a combination of birth-death Markov chains, Mean Value Analysis, and hierarchical modeling. The software server architectures considered in this paper cover a wide range of situations found in modern server-oriented systems. The modeling approach presented here takes into account both software and hardware contention in single and multiclass situations.

References

- [1] L. Kleinrock, *Queueing Systems Volume I: Theory*, John Wiley & Sons, New York, NY, 1975.
- [2] E. Lazowska, J. Zahorhan, G.S. Graham, and K.C. Sevcik, *Quantitative System Performance: computer system analysis using queuing network models*, Prentice Hall, Upper Saddle River, 1984.
- [3] D.A. Menascé, V.A.F. Almeida, and L.W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*, Prentice Hall, Upper Saddle River, 2004.
- [4] M. Reiser and S. Lavenberg, "Mean-value analysis of closed multi-chain queuing networks," *J. ACM*, vol. 27, no. 2, 1980, pp. 643–673.
- [5] P. Schweitzer, "Approximate analysis of multiclass closed network of queues," *Proc. Intl. Conf. Stochastic Control and Optimization*, Amsterdam, 1979.

Iteration	\bar{n}_1^{exec}	\bar{n}_1^{syst}	δ_1^{exec}	M_1^*	$R_{0,1}$	% relative error 1	\bar{n}_2^{exec}	\bar{n}_2^{syst}	δ_2^{exec}	M_2^*	$R_{0,2}$	% relative error 2
0	3.130	3.130	2.164	7.836	-	-	2.164	2.164	2.978	7.022	-	-
1	2.978	3.256	2.148	7.852	0.1302	5.1	2.148	2.132	3.077	6.923	0.1066	0.8
2	3.077	3.245	2.160	7.840	0.1298	3.2	2.160	2.216	3.085	6.915	0.1108	0.6
3	3.085	3.253	2.163	7.837	0.1301	0.3	2.163	2.219	3.085	6.915	0.1110	0.2

Table 2: Results for multiclass unlimited thread queue case.

Iteration step:

for every class $r = 1, 2, \dots, R$ do

begin

$$\delta_r^{\text{exec}} = \sum_{r' \neq r} \bar{n}_{r'}^{\text{exec}}$$

$$\delta_r^{\text{syst}} = \sum_{r' \neq r} \bar{n}_{r'}^{\text{syst}}$$

/* Adjust the values of M and N */

$$M^* = M - \delta_r^{\text{exec}}$$

$$N^* = N - \delta_r^{\text{syst}}$$

for $n_r = 1, 2, \dots, \lfloor M^* \rfloor$ do

Call aMVA to compute $X_r(\bar{n}_1^{\text{exec}}, \bar{n}_2^{\text{exec}}, \dots, n_r, \dots, \bar{n}_R^{\text{exec}})$

if M^* is not an integer then

Call aMVA to compute $X_r(\bar{n}_1^{\text{exec}}, \bar{n}_2^{\text{exec}}, \dots, M^*, \dots, \bar{n}_R^{\text{exec}})$

define a single class generalized birth-death model with $\lambda = \lambda_r$ and death rates:

$$\mu_r(k) = X_r(\bar{n}_1^{\text{exec}}, \bar{n}_2^{\text{exec}}, \dots, k, \dots, \bar{n}_R^{\text{exec}}), \text{ for } 1 \leq k \leq \lfloor M^* \rfloor, \text{ and}$$

$$\mu_r(k) = X_r(\bar{n}_1^{\text{exec}}, \bar{n}_2^{\text{exec}}, \dots, M^*, \dots, \bar{n}_R^{\text{exec}}) \text{ for } \lfloor M^* \rfloor < k \leq \lfloor N^* \rfloor.$$

compute the probability distribution, P_k , according to Section 4, as

$$P_k = P_0 \times \lambda_r^k / \beta(k), \text{ for } 1 \leq k \leq \lfloor M^* \rfloor, \text{ and}$$

$$P_k = P_0 \times \rho^k \times \mu_r(M^*)^{\lfloor M^* \rfloor} / \beta(\lfloor M^* \rfloor), \text{ for } \lfloor M^* \rfloor < k \leq \lfloor N^* \rfloor$$

where $\rho = \lambda_r / \mu_r(M^*)$, $\beta(k) = \mu_r(1) \times \dots \times \mu_r(k)$, and

$$P_0 = \left[1 + \sum_{k=1}^{\lfloor M^* \rfloor} \lambda_r^k / \beta(k) + \frac{\rho \times \lambda_r^{\lfloor M^* \rfloor} \times (1 - \rho^{\lfloor N^* \rfloor - \lfloor M^* \rfloor})}{\beta(\lfloor M^* \rfloor) \times (1 - \rho)} \right]^{-1}$$

Compute the performance metrics as:

average number of requests in the system:

$$\bar{n}_r^{\text{syst}} = \sum_{k=1}^{\lfloor N^* \rfloor} k \times P_k$$

average throughput:

$$\bar{X}_r = \sum_{k=1}^{\lfloor N^* \rfloor} P_k \times \mu_r(k)$$

probability of rejection:

$$P_{r,\text{rej}} = P_{\lfloor N^* \rfloor}$$

average response time:

$$R_{0,r} = \bar{n}_r^{\text{syst}} / \bar{X}_r$$

average number of requests in execution:

$$\bar{n}_r^{\text{exec}} = \sum_{k=1}^{\lfloor M^* \rfloor - 1} k \times P_k + M^* \times [1 - \sum_{k=0}^{\lfloor M^* \rfloor - 1} P_k]$$

end

Convergence test step: Repeat the iteration step until successive values of \bar{n}_r^{exec} are sufficiently close for all classes

Compute final metrics: Return the performance metrics as computed in the last iteration

Figure 9: Iterative Phase of the Performance Model for a Multiclass Multithreaded Server with Limited Queue.