Two-Level Iterative Queuing Modeling of Software Contention

Daniel A. Menascé
Department of Computer Science
George Mason University
Fairfax, VA 22030
menasce@cs.gmu.edu

Abstract

Being able to model contention for software resources (e.g., a critical section or database lock) is paramount to building performance models that capture all aspects of the delay encountered by a process as it executes. Several methods have been offered for dealing with software contention and with message blocking in client-server systems. This paper presents a general, straightforward, easy to understand and implement, approach to modeling software contention using queuing networks. The approach, called SQN-HQN, consists of a two-level iterative process. Two queuing networks are considered: one represents software resources (SQN) and the other hardware resources (HQN). Multiclass models are allowed and any solution technique—exact or approximate—can be used at any of the levels. This technique falls in the general category of fixed-point approximate models and is similar in nature to other approaches. The main difference lies in its simplicity. The process converges very fast in the examples examined. The results were validated against global balance equation solutions and are very accurate.

1. Introduction

The response time of a transaction or request submitted to a computer system can be broken down into three components. The first is the total service time, i.e., the total time spent by transactions obtaining service from the various physical resources such as processors, disks, and networks. The second component is the total time spent waiting to use a physical resource. Finally, the third component (*software contention*), is the time spent to access a software resource such as a non-reentrant software module, a software thread, or a database lock. It is important to take into account the effect of software contention when estimating the total response time.

Agrawal and Buzen [1] present an approximate iterative technique to model serialization delays that employs a conventional queuing network (QN) comprised of servers that represent actual processors and I/O devices plus additional aggregate servers that represent serialized processing activity. The technique iteratively determines the values of the service times at all devices by computing adjustment factors used to elongate the original service times. The formulas provided for computing these adjustment factors are neither simple nor intuitive. The formulation is presented for single class models only. The extension to multiple classes is said to follow the same principles but is acknowledged to be computationally difficult [1]. The work in [21] is a two-level model of serialization delays. At the lower level, a QN that represents the physical resources is used to compute transition rates for a higher level Markov Chain model. This formulation does not generalize to multiple classes and does not deal with other types of software contention issues. The approach taken in [7] for analyzing QNs with serialization delays consists in transforming the network to one with population constraints and using an approximation technique to solve the latter problem. Other analytic work on specific software contention situations can be found in [10] and [16].

Multilayer client-server queuing network models were introduced in [15] to handle synchronous and asynchronous exchange of messages between clients and servers. The problem of modeling CORBA based distributed systems was addressed in [9] through the use of QNs with simultaneous resource possession. These networks are solved through decomposition into a set of auxiliary product-form QNs and iteration between these networks.

A very important body of work aimed at estimating software contention issues is the Method of Layers (MOL), also called Layered Queuing Networks (LQN) [17, 23]. Also related and very relevant is the work on stochastic rendezvous networks [14, 22]. Layered queuing networks were extended to handle parallel tasks in [4]. The LQN approach while, in principle, similar to the technique presented here differs in ways that will be explained in the concluding remarks, after our technique has been presented.

Proceedings of the 10th IEEE Int'l Symp. on Modeling, Analysis, & Simulation of Computer & Telecommunications Systems (MASCOTS'02)
1526-7539/02 $17.00 © 2002 **IEEE**

The main contribution of this paper is that it provides a simple analytical modeling technique for estimating software contention delays. The technique, called SQN-HQN, consists of a two-layer queuing network model. Software resources are modeled by a software queuing network (SQN) and physical resources by a hardware queuing network (HQN). The effect of software contention on the physical QN and the effect of physical resource contention on the software QN requires an iterative technique to solve this dependency. The technique relies on existing solution techniques for open, closed, multiclass queuing networks and can be used with both product-form QNs and approximations [2] used to handle situations such as simultaneous resource possession [2, 8, 12] and priorities [2, 3, 18]. Multiservers can be handled by using load-dependent servers [11, 13] or approximations [19]. The basic idea developed in this paper was first proposed in [11] but no algorithm and no validation was presented there. While the other techniques referenced earlier can also handle the situations addressed by our approach, we believe that the strength of our method lies in its simplicity. Anyone who understands how to solve QNs can use the method to estimate software contention.

The rest of the paper is organized as follows. Section 2 illustrates the main rationale behind the technique through a simple motivating example in which processes compete for a single critical section and execute non-critical section code after they complete the critical section. A single class algorithm is also presented in that section. The next section presents a more complex example and compares our results with those obtained with other techniques. Section four shows how the technique presented in section 2 can be extended to model other cases. Section 5 presents the multiclass algorithm and section 6 presents some concluding remarks.

2. A Simple Example - Contention for a Critical Section

Consider the case of processes P_1 P that alternate between executing non-critical section and critical section code. Any number of processes can be concurrently executing their non-critical code. But, only one of them can be executing the critical section code. If a process attempts to enter its critical section code while another process is inside the critical section, the attempting process is put to sleep at the queue associated with the semaphore that controls access to the critical section. Let us assume in this simple example that the only physical device used by all processes during the execution of the critical and non-critical section code is the CPU and that the CPU scheduling discipline is processor-sharing.

If we just consider the software phases of a process execution, we can depict a process by a *software queuing network* (SQN) as in the top part of Fig. 1. The SQN has two resources (software modules). One is a delay-resource (illustrated as a rectangle) that corresponds to the non-critical section code; there is no queuing for a software resource during this phase. The other software resource is a queuing resource, which corresponds to the critical section code.

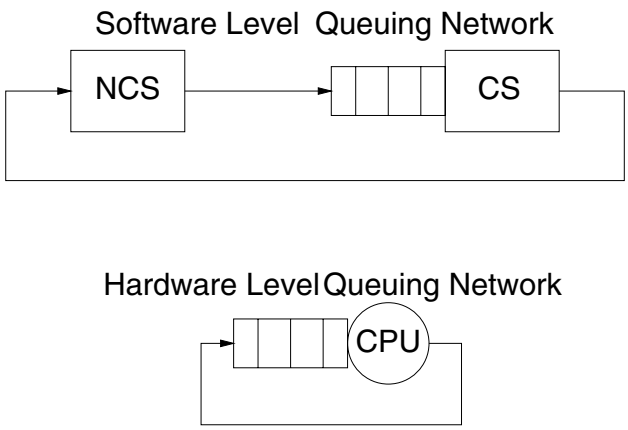

Figure 1. Software and hardware queuing networks for the critical section example

While a process—a customer in the SQN—is using the software resources in the SQN, it is also using or waiting to use physical resources (e.g., CPUs and disks). Delay resources are used in the SQN to represent software resources for which there is no software contention. The queuing network associated with the physical resources, the *hardware queuing network* (HQN), is shown in the bottom part of Fig. 1. Customers in the HQN are processes using the physical resources as a result of the execution of software modules. The time spent at the NCS and CS resources in the SQN depends on the contention for access to the physical resources, the CPU in this case. Also, the number of processes contending for the CPU, i.e., the number of processes in the HQN, is equal to the number of concurrent processes that are not blocked waiting for entry to the critical section. The blocked processes are sleeping and are therefore not present in any HQN queue. Therefore, the customer population in the HQN is equal to $-$ where is the number of processes blocked for a software resource.

We now define some notation before we proceed with an explanation on how contention for access to software resources can be computed. Let

- $D_{\ i}$: software-hardware service demand, i.e., total service time of a process executing software module when using physical resource i in the HQN. The superscript "sh" indicates that these service demands are

related to the mapping between the software and hardware QNs. For example, D is the total CPU time for the execution of the non-critical section code. This time does not include the time spent waiting to use the CPU while executing non-critical section code.

- D : software service demand, i.e., total service time to execute module in the SQN. The superscript "s" indicates that this service demand relates to the software QN. For example, D is the total service time to execute the non-critical section code. The service demand D is the sum of all service times at all physical devices during the execution of module . Thus,

$$D = \sum_i D_{\ i}. \quad (1)$$

- D_i : hardware service demand, i.e., total service time of a process at physical resource i in the hardware QN. For example, D is the total service time of a process at the CPU. This time, is the sum of the service demands due to the execution of all modules of the process. Thus,

$$D_i = \sum D_{\ i}. \quad (2)$$

For example, $D = D \quad D$.

- $'_i(\ )$: residence time, i.e., total time spent by a process at physical resource i, waiting for or receiving service, when there are processes at the HQN.

We now show the iterative algorithm used to estimate the software contention time and to compute all performance metrics (e.g., response time and throughput).

The inputs to the SQN-HQN algorithm are the service demands $D_{\ i}$ and the number of processes. The algorithm iterates between solving the SQN and the HQN. Figure 2 illustrates the relationship between the SQN and HQN models. The SQN model receives as input the number of processes and the software service demands and produces the number of processes blocked for software resources as output. The HQN model takes as input a number of customers , which is the original number of processes minus the number of processes blocked for software resources, and the hardware service demands. The output of the HQN model is the set of residence times at each physical device. These residence times are used to adjust (see step 5 of the algorithm) the software service demands for the SQN model. The iteration starts by solving the SQN assuming zero contention for physical resources. The algorithm iterates until successive values of the number, , of processes blocked for software contention are sufficiently close.

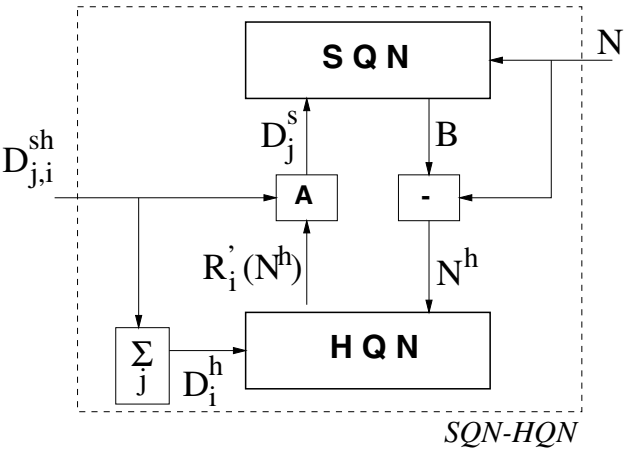

Figure 2. SQN-HQN scheme.

- Step 1 - Initialization:

$$D \quad \sum_i D_{\ i} \quad (3)$$

$$D_i \quad \sum D_{\ i} \quad (4)$$

$$^0 \quad 0 \quad (5)$$

$$k \quad 1 \quad (6)$$

- Step 2 - Solve the SQN with D as service demands and as customer population.

- Step 3 - Compute the average number of blocked processes ^{k}. In the case of our example, this is equal to the average number of processes waiting in the queue for the CS resource. So, $^k = \overline{\ } \ -$, where $\overline{\ }$ is the average number of processes at resource CS in the SQN and is the utilization of resource CS. In general,

$$^k = \sum \quad (7)$$

where is the average number of processes in the waiting line for software resource .

- Step 4 - Solve the HQN with D_i as service demands and $=$ $-$ ^{k} as customer population. Note that the solution to a QN with a non-integer customer population can be obtained using Schweitzer's approximation [20].

- Step 5 - Adjust the service demands at the SQN to account for contention at the physical resources. So,

$$D \quad \sum_i \frac{D_{\ i}}{D_i} \times \ '_i(\quad). \quad (8)$$

So, for our example we have

$$D \quad \frac{D}{D} \times \quad ' \quad (\quad) \qquad (9)$$

$$D \quad \frac{D}{D} \times \quad ' \quad (\quad). \qquad (10)$$

- Step 6 (Convergence Test): If $(\ ^{k} - \ ^{k-1})/ \ ^{k}$ then $k \quad k \quad 1$ and go to step 2.

We now give a justification for Step 5 of the above algorithm using our critical section example. The residence time equation for MVA applied to the HQN is

$$' \quad (\quad) = D \quad [1 \quad - \quad (\quad - 1)]. \qquad (11)$$

But,

$$D \quad = D \qquad D \quad . \qquad (12)$$

So, using Eqs. (11) and (12), we get

$$\begin{aligned} ' \quad (\quad) \quad = \quad & D \qquad [1 \quad - \quad (\quad - 1)] \\ & D \qquad [1 \quad - \quad (\quad - 1)]. \end{aligned} \qquad (13)$$

The first term of the right-hand side of Eq. (13) is the total time (waiting + service) spent at the CPU by a process while executing non-critical section code and the second term is the total time spent at the CPU by a process while executing the critical section code. So, for example, using Eqs. (11) and (13) we can write the total time spent at the CPU while executing the non-critical section code as

$$\frac{D}{D} \times \quad ' \quad (\quad). \qquad (14)$$

Figure 3 shows the percentage of the total response time spent waiting to enter the critical section as a function of the number of concurrent processes. The service demands used to obtain the graph are: $D \qquad = 0.2$ sec and $D \qquad = 0.1$ sec. As it can be seen, as the concurrency level increases, contention for software resources dominates the response time of a process.

Figure 4 shows the percentage of the total response time spent waiting to enter the critical section as a function of the ratio between non-critical section and critical section time, i.e., as a function of the ratio $= D \qquad /D \qquad$ for different values of the number of concurrent processes. As expected, as increases, a smaller fraction of the response time is spent waiting to enter critical section. Consider $= 5$ for example. When $= 1$, 62% of the response time is spent waiting to enter the critical section. When is multiplied by 10, the fraction of time spent waiting to enter critical section falls to 4.6%.

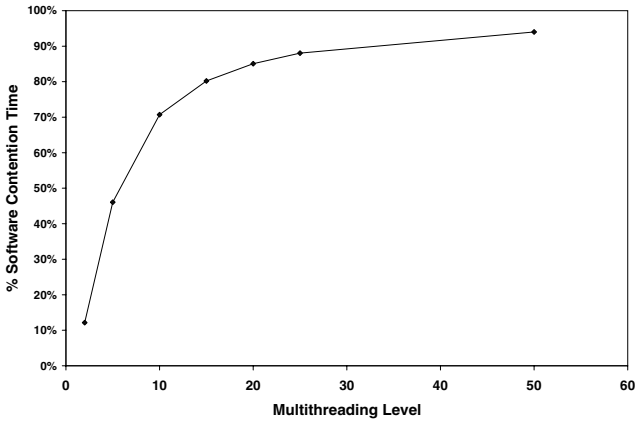

Figure 3. Percent of software contention time vs. multithreading level in the critical section example

The solutions obtained for this example match exactly the results obtained from solving global balance equations for $= 2$ (see appendix A for this solution). Other examples in this paper show comparisons with global balance equation solutions published elsewhere.

3. A More Complex Example

In this section we apply the algorithm presented in the previous section to an example that appears in [1]. We use this example for two reasons. First, because the example is more complex than the one presented before. It has two critical sections at the software level (see top portion of Fig. 5). The hardware QN is composed of one CPU and three disks (see bottom part of Fig. 5). Processes use both CPU and disk while in the critical section. The second reason for using the example in [1] is that global balance solutions are provided in that paper. So, we can compare the results provided by our technique with exact results.

The service demands (i.e, the values of $D \ _{i}$) are given in Table 1 and are the same as in [1]. The last row shows the values of D and the last column contains the values of D_i.

Table 2 shows the results (i.e., throughputs) obtained with our technique (called SQN-HQN) and with global balance equations, as reported in [1], for eight values of the number of concurrent processes . The last three columns of the table show the absolute % relative error obtained with our technique and with those of [1] and [7] relative to the global balance equations solution. The error of our approach is very small and does not exceed 3.05% in this example; for 1, it is always smaller than the one reported in [1]. The error obtained with our method is higher

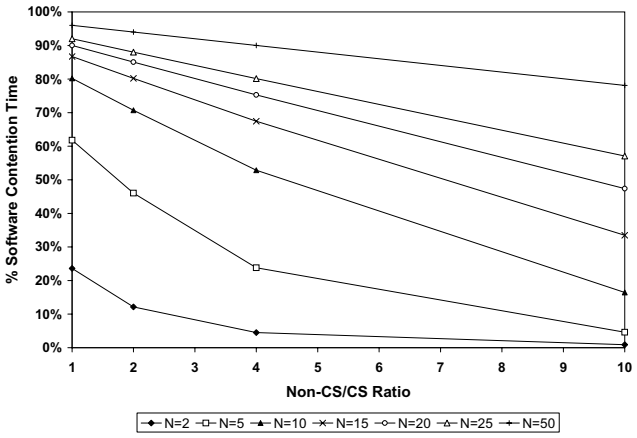

Figure 4. Percent of software contention time vs. ratio between non-CS and CS Time for different values of

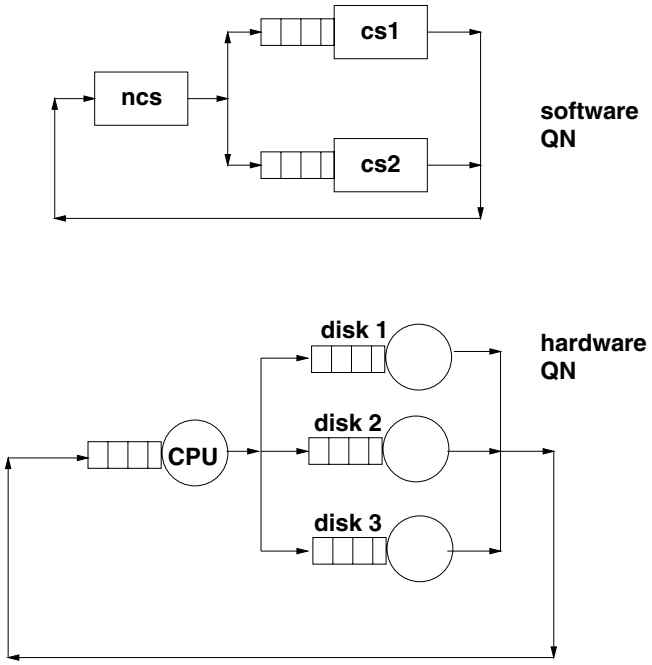

Figure 5. Software and hardware queuing networks for the example in [1]

Table 1. Service demands (in sec) for the example in [1]

	Software Module			Hardware
Device	NCS	CS 1	CS 2	Demands
CPU	0.2000	0.0600	0.0808	0.3408
Disk 1	0.0560	0.0576	0.000	0.1136
Disk 2	0.0360	0.0000	0.1212	0.1572
Disk 3	0.0360	0.0000	0.0000	0.0360
Software Demands	0.3280	0.1176	0.2020	

than the one for [7] for 4 but our method is much simpler to implement. It should also be noticed that the results obtained with the SQN-HQN technique are consistently pessimistic.

4. Other Cases

In this section we explain how the algorithm presented in Section 2 can be extended to cover other situations. The first is the case in which a customer in the SQN spends time not directly associated with the execution of a software module (e.g., think times or network time). The second case discusses how open QNs can be used to represent SQNs.

4.1 Modeling Non-Software Resources

There are cases in which there is no correspondence between a resource at the hardware and the software layers. For example, Fig. 6 shows the case in which user think time as well as network waiting and transmission time need to be modeled. In these cases, we need to incorporate these two resources—a delay device for the think time and a queuing device for the network—in both the SQN and HQN networks. We call them *non-software* resources. Note that we depict software resources by rectangles in the SQN and all other resources by circles. The computation of ^{k} in Step 3 of the algorithm presented in the previous section only includes processes blocked for software resources.

The service demands for the example of Fig. 6 are shown in Table 3. All demands except for the think time and network are the same as in the previous example. The client think time is 10 seconds and the network service demand of a request is 0.012 sec.

Figure 7 shows the response time and the fraction of client requests blocked for software contention as a function of the total number of clients . As it can be seen, as the number of clients increases, the response time increases as expected. A more important observation is the significant increase of the percentage of client requests blocked for a software resource (critical section 1 or 2). For example, when the number of clients is equal to 35, close to 30% of the clients are waiting for a software resource.

Table 2. Throughput(processes/sec) for example in [1] (GBS = Global Balance Solution, SQN-HQN= technique presented in this paper, ASM = algorithm in [1], and ASPA = algorithm in [7])

N	SQN-HQN	GBS	Absolute % Relative Error		
			SQN-HQN	ASM	ASPA
1	1.544	1.54	0.27	0.00	-
2	2.088	2.11	1.06	4.60	-
3	2.317	2.37	2.22	5.89	4.2
4	2.428	2.49	2.49	5.99	2.8
5	2.487	2.56	2.86	5.87	2.0
6	2.521	2.60	3.05	5.78	1.5
7	2.541	2.62	3.00	5.75	1.5
8	2.555	2.63	2.86	5.77	1.1

4.2. Open QN at the Software Level

The technique presented in this paper can be extended to include the case in which the SQN is modeled as an open queuing network as illustrated in Fig. 8. In that case, processes arrive at a rate of λ requests/sec. The rest of the SQN and the HQN are the same as in the example of Fig. 5 and the service demands are as in Table 1.

Steps 2 and 4 of the algorithm described in Section 2 should be replaced by the following to contemplate the case of open SQNs.

- new Step 2 - Solve the SQN with D as service demands and λ as arrival rate and obtain $\overline{\ }$ the average number of customers in the SQN.
- new Step 4 - Solve the HQN with D_i as service demands and $= \overline{\ } - {}^{k}$ as customer population. Note that the solution to a QN with a non-integer customer population can be obtained using Schweitzer's approximation [20].

Table 3. Service demands (in sec) for the example in Fig. 6

	Software Module					Hardware
Device	NCS	CS 1	CS 2	Think Time	Network	Demands
CPU	0.2000	0.0600	0.0808	0.000	0.000	0.3408
Disk 1	0.0560	0.0576	0.0000	0.000	0.000	0.1136
Disk 2	0.0360	0.0000	0.1212	0.000	0.000	0.1572
Disk 3	0.0360	0.0000	0.0000	0.000	0.000	0.0360
Think Time	0.0000	0.0000	0.0000	10.000	0.000	10.0000
Network	0.0000	0.0000	0.0000	0.000	0.012	0.0120
Software Demands	0.3280	0.1176	0.2020	10.000	0.012	

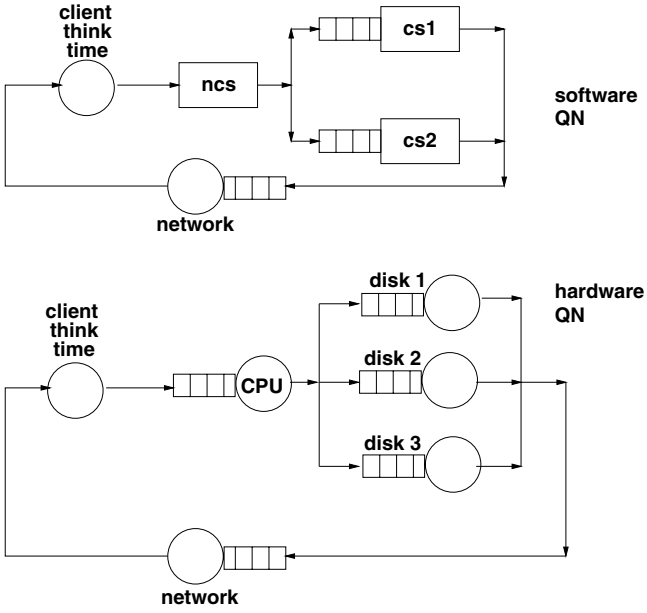

Figure 6. Example with client think time and network

Table 4 contains results of running the modified algorithm for the case of $\lambda = 3$ processes/sec. The table shows the results of the first 10 iterations. Column 2 ( ) is the customer population used to solve the HQN at each iteration. The next column shows the average response time . Columns 4 and 5 show the average number of requests in the SQN and the average number of blocked requests in the CS1 and CS2 queues, respectively. The last three columns show the modified service demands for the SQN at each step. Note that the values for iteration 0 are the original values of these demands. The relative error in B is 0.123% at the 10-th iteration.

Table 4. Results of the first 10 iterations for the open SQN example

					Modified SQN Demands		
Iter				B	NCS	CS1	CS2
0	-				0.3280	0.1176	0.2020
1	1.295	0.821	1.641	0.346	0.3662	0.1302	0.2235
2	1.440	0.946	1.893	0.453	0.3858	0.1365	0.2342
3	1.513	1.014	2.028	0.515	0.3958	0.1397	0.2396
4	1.550	1.050	2.100	0.549	0.4010	0.1414	0.2424
5	1.570	1.069	2.137	0.568	0.4037	0.1423	0.2438
6	1.580	1.079	2.157	0.577	0.4051	0.1427	0.2446
7	1.585	1.084	2.167	0.582	0.4059	0.1430	0.2450
8	1.588	1.086	2.173	0.585	0.4062	0.1431	0.2452
9	1.589	1.088	2.175	0.587	0.4064	0.1432	0.2453
10	1.590	1.088	2.177	0.587			

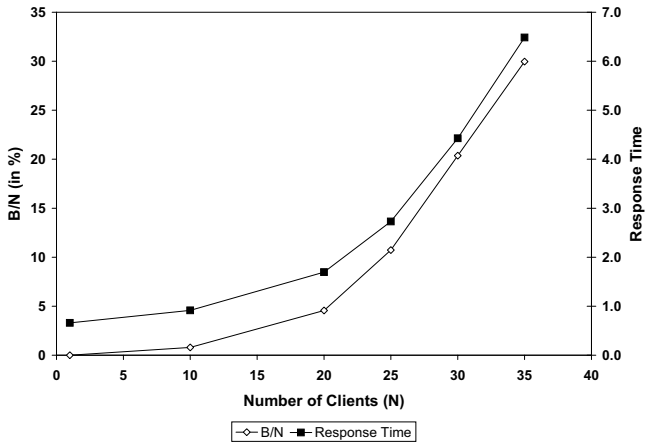

Figure 7. Response time and percentage of clients blocked for software contention

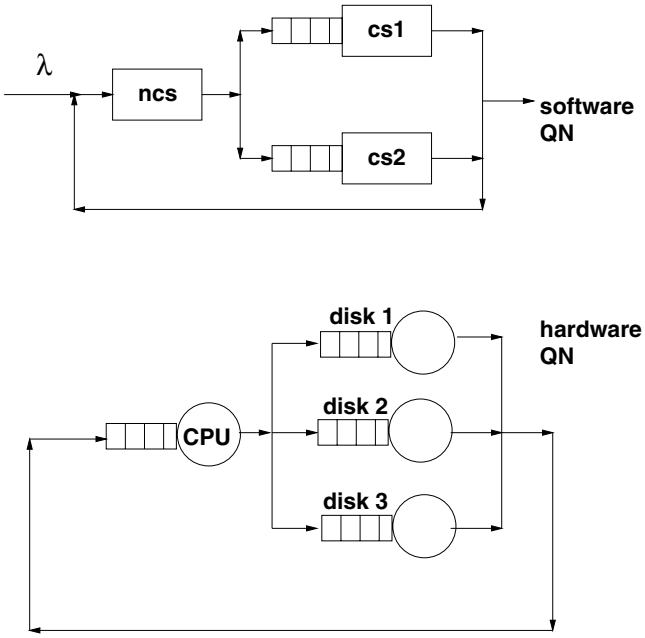

Figure 8. Open QN at the software level

5. The Multiclass Algorithm

The algorithm presented in Section 2 generalizes in a straightforward manner to multiple classes. We define here the generalized notation and present the multiclass algorithm. Let

- : number of classes in the SQN and in the HQN.
- $= (\ _1$): population vector for the SQN. is the multithreading level for class r.
- $= (\ _1$): population vector for the HQN. is the number of customers in class r.
- $= (\ _1$): vector of number of processes blocked, i.e., waiting for a software resource. is the number of class r processes blocked for a software resource.
- $D\ _i$: software-hardware service demand, i.e., total service time at physical resource i of a class r software module .
- D : software service demand, i.e., total service time to execute module of class r in the SQN. The service demand D is the sum of all service times at all physical devices during the execution of module . Thus,

$$D \quad = \sum_i D\ _i \ . \tag{15}$$

- D_i : hardware service demand, i.e., total service time at physical resource i for class r in the hardware QN. This time, is the sum of the service demands due to the execution of all modules of a process. Thus,

$$D_i \quad = \sum D\ _i \ . \tag{16}$$

- $_i'$ ( ): residence time, i.e., total time spent by a class r process at resource i, waiting for or receiving service, when the customer population at the HQN is given by the vector .

We now show the iterative algorithm used to estimate the software contention time and to compute all performance metrics (e.g., response time and throughput). The inputs to the algorithms are the service demands $D\ _i$ and the vector . The algorithm iterates until successive values of are sufficiently close.

- Step 1 - Initialization:

$$D \qquad \sum_i D\ _i \qquad r = 1 \tag{17}$$

$$D_i \qquad \sum D\ _i \qquad r = 1 \tag{18}$$

$$0 \tag{19}$$

$$k \qquad 1 \tag{20}$$

- Step 2 - Solve the SQN with D as service demands and as customer population.

- Step 3 - Compute the average number of blocked processes per class as

$$^{k} = (\sum \quad _{1} \qquad \sum \qquad \sum \qquad) \quad (21)$$

where $r = 1$, is the average number of class r processes in the waiting line for software resource .

- Step 4 - Solve the HQN with D_i as service demands and population vector $= \quad - \ ^{k}$ as customer population. Note that the solution to a QN with a non-integer customer population can be obtained using Schweitzer's approximation [20]. The solution to the HQN provides the residence time values $_{i}^{\prime}$ ( ) for all devices i and classes r.

- Step 5 - Adjust the service demands at the SQN to account for contention at the physical resources. So,

$$D \qquad \sum_{i} \frac{D \ _{i}}{D_i} \times \ _{i}^{\prime} (\quad). \qquad (22)$$

- Step 6 (Convergence Test):
If ($^{k} - \ ^{k-1}$)/ ^{k} then $k \quad k \quad 1$ and go to step 2.

The same considerations discussed in section 4 can be applied to the multiclass case. Situations that require modeling a queue for a multithreaded software server can be considered by including a multiserver queue (see Fig. 9) in the SQN and using approximation techniques [19] to solve QNs with multiservers or QNs with load-dependent servers [11, 13].

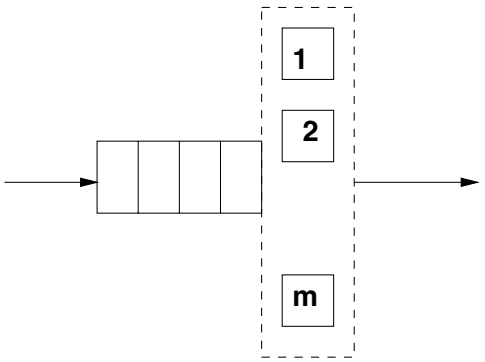

Figure 9. A multithreaded server process

6. Concluding Remarks

This paper presented a simple technique to estimate software contention using analytic models. The approach uses well-known hierarchical modeling methods solved through iteration. The technique consists of two queuing networks. The top level—the software queuing network (SQN)—has servers associated with software resources. A class of customers represents processes with similar behavior in their use of software resources and use of the underlying physical resources while using software resources. Mutiple classes of customers are allowed. Software resources that do not generate any software contention are modeled as delay resources in the SQN and contention for software resources is represented by queuing resources. These may have a single server (as in the case of a critical section or a serialization delay) or multiple servers as in the case of a multithreaded process. The SQN may be open or closed. Any known technique, exact or approximate, may be used to solve the SQN. Situations that involve processes that request service more than once (multi-phased processes) from the same software resource and have significantly different service times at each phase, can be modeled using class-switching QNs as done in [5, 6].

The bottom-level QN represents the hardware resources. Service demands at the software level are iteratively adjusted to account for contention at the physical level. Again, any exact or approximate technique can be used to solve the hardware QN, which is modeled as a closed QN.

Our approach differs from the LQN method [17] in that i) only two layers are considered, ii) any QN solution technique can be used at the SQN and HQN without any need to adapt specific residence time equations as done in [17], and iii) open and closed QNs are allowed at the SQN. The hope is that by providing a straightforward and easy to understand method for modeling software contention, those who are familiar with solution methods for multiclass QNs will be able to readily model software contention without the need to use new types of QNs. While simple, the technique presented here was validated and is general enough to be applied to multiclass situations.

Appendix: A Global Balance Equations Solution

In this appendix we present a Markov Chain-based solution to the critical section problem of Section 2 for the special case of two identical processes. The rate of completion of the NCS phase is μ if only one process is using the CPU and the rate of completion of the CS phase is μ if only one process is active. So, $\mu \ = 1/D$ and $\mu \ = 1/D$. We can depict this system as composed of the five states shown in Fig. 10: (ncs, ncs), (ncs, cs), (cs,

ncs), (sl,cs), and (cs, sl), where $(i \quad)$ indicates that process 1 is in state i and process 2 in state . The state of a process can be ncs (executing non-critical section code), cs (executing critical section code), and sl (sleeping - waiting to enter critical section). We are assuming processor sharing queuing discipline at the CPU. So, when both processes are active—states (ncs, ncs), (ncs, cs), and (cs, ncs)—the rate of completion of critical section and non-critical section code by each process is half the rate one would see if only one process were using the CPU.

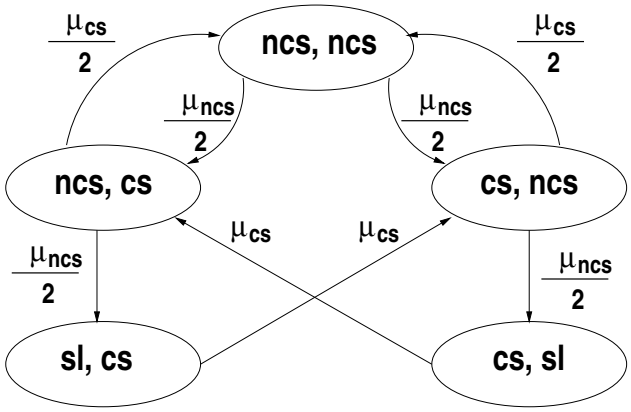

Figure 10. State transition diagram for critical section with two processes

The global balance equations are:

$$
\begin{aligned}
(\mu \quad /2) \times \qquad &= \mu \quad \times \\
(\mu \quad /2) \times \qquad &= \mu \quad \times \\
(\mu \quad /2) \times \qquad \mu \quad \times \qquad &= \frac{\mu \quad \mu}{2} \times \\
(\mu \quad /2) \times \qquad \mu \quad \times \qquad &= \frac{\mu \quad \mu}{2} \times \\
&= 1 \qquad (23)
\end{aligned}
$$

The solution to the above system of linear equations is:

$$= \quad \frac{\mu}{\mu \quad \mu}^{2} \qquad (24)$$

$$= \quad = \quad \frac{\mu}{\mu} \qquad (25)$$

$$= \quad = \quad \frac{1}{2}. \frac{\mu}{\mu}^{2} \qquad (26)$$

One can now compute the system throughput X by computing the rate at which each process completes its non-critical section code. This is given by

$$X = (\mu \quad /2) \times (\qquad) \times 2 = \frac{\mu \quad \times \mu}{\mu \quad \mu}. \qquad (27)$$

References

[1] S. C. Agrawal and J. P. Buzen, "The Aggregate Server Method for Analyzing Serialization Delays in Computer Systems," *ACM Tr. Comp. Sys.*, Vol. 1, No. 2, May 1983, pp. 116–143.

[2] S. C. Agrawal, "Metamodeling: A study of Approximations in Queuing Models," MIT Press, 1985.

[3] V.A.F. Almeida and D. A. Menascé, "Approximate Modeling of CPU Preemptive Resume Priority Scheduling Using Operational Analysis," Proc. 10th European Computer Manufactures Association (ECOMA) Conf. on Computer Measurement, Munich, Germany, Oct. 12-15, 1982.

[4] G. Franks and C. M. Woodside, "Performance of Multi-Level Client-Server Systems with Parallel Service Operations," Proc. First Intl. Workshop on Software and Performance (WOSP'98), Santa Fe, NM, Oct. 1998, pp. 120–130.

[5] A. Harbitter and D. A. Menascé, "The Performance of Public Key Enabled Kerberos Authentication in Mobile Computing Applications," Eighth ACM Conf. Computer and Communications Security (CCS-8), Philadelphia, PA, Nov. 5-8, 2001.

[6] A. Harbitter and D. A. Menascé, "Performance of Public Key-Enabled Kerberos Authentication in Large Networks," Proc. 2001 IEEE Symp. on Security and Privacy, Oakland, CA, May 13-16, 2001.

[7] P. A. Jacobson and E. D. Lazowska, A Reduction Technique for Evaluating Queuing Networks with Serialization Delays," *Performance'83*, eds. A. K. Agrawal and S. K. Tripathi, North-Holland Publishing Company, 1983, pp. 45–59.

[8] P. A. Jacobson and E. D. Lazowska, "Analyzing Queuing Networks with Simultaneous Resource Possession," *Comm. ACM*, 25, 2, Feb. 1982, pp. 142–151.

[9] P. Kahkipuro, "Performance Modeling Framework for CORBA Based Distributed Systems," Ph.D. Dissertation, Technical Report A-2000-3, Department of Computer Science, University of Helsinki, Finland, 2000.

[10] C. M. Lladó and P. G. Harrison, "Performance Evaluation of an Enterprise JavaBean Server Implementation," Proc. Second Intl. Workshop on Software and Performance, Ottawa, Canada, Sept. 17-20, 2000, pp. 180–188.

[11] E. D. Lazowska, J. Zahorjan, G. S. Graham, K. C. Sevcik, "Quantitative System Performance," Prentice Hall, 1984.

[12] D. A. Menascé, O. Pentakalos and Y. Yesha, "An Analytic Model of Hierarchical Mass Storage Systems with Network-Attached Storage Devices," Proc. 1996 ACM Sigmetrics Conference Philadelphia, PA, May 1996.

[13] D. A. Menascé and V. A. F. Almeida, "Capacity Planning and Performance Modeling: from mainframes to client-server systems," Prentice Hall, Upper Saddle River, NJ, 1994.

[14] D. Petriu, H. Amer, S. Majumdar, I. Abdull-Fatah, "Using Analytic Models for Predicting Middleware Performance," Proc. Second Intl. Workshop on Software and Performance, Ottawa, Canada, Sept. 17-20, 2000, pp. 189–194.

[15] S. Ramesh and H. G. Perros, "A Multilayer Client-Server Queuing Network Model with Synchronous and Asynchronous Messages," *IEEE Tr. Software Eng.*, Vol. 26, no. 11, Nov. 2000, pp. 1086–1100.

[16] P. Reeser and R. Hariharan, "Analytic Model of Web Servers in Distributed Environments," Proc. Second International Workshop on Software and Performance, Ottawa, Canada, Sept. 17-20, 2000, pp. 158–167.

[17] J. A. Rolia and K. C. Sevcik, "The Method of Layers," *IEEE Tr. Software Eng.*, vol. 21, no. 8, 1995, pp. 689–700.

[18] K. C. Sevcik, "Priority Scheduling Disciplines in Queuing Network Models of Computer Systems," Proc. IFIP Congress 77, North-Holland Publishing Co., Amsterdam, 1977, pp. 565–570.

[19] A. Seidmann, P. Schweitzer and S. Shalev-Oren, "Computerized Closed Queueing Network Models of Flexible Manufacturing Systems," *Large Scale Syst. J.*, North Holland, vol. 12, pp. 91–107, 1987.

[20] P. Schweitzer, Approximate analysis of multiclass closed network of queues, in *International Conference on Stochastic Control and Optimization*, Amsterdam, 1979.

[21] Thomasian, A., "Queuing Network Models to Estimate Serialization Delays in Computer Systems," *Performance'83*, eds. A. K. Agrawal and S. K. Tripathi, North-Holland Publishing Company, 1983, pp. 61–81.

[22] C. M. Woodside, J. E. Neilson, D. C. Petriu, and S. Majumdar, "The Stochastic Rendez-vous Network Model for Performance of Synchronous Client-Server-like Distributed Software," *IEEE Tr. Computers*, vol. 44, no. 1, 1995.

[23] C. M. Woodside, "An Active-Server Model for the Performance of Parallel Programs Written Using Rendezvous," *J. Systems and Software*, 1986, pp. 125–131.