Fast Approximate Solution of Multiprogramming Models

Alexandre Brandwajn

Amdahl Corporation

Sunnyvale, CA 94086

ABSTRACT

Queueing network models of computer systems with multiprogramming constraints generally do not possess a product-form solution in the sense of Jackson. Therefore, one is usually led to consider approximation techniques when dealing with such models. Equivalence and decomposition is one way of approaching their solution. With multiple job classes, the equivalent network may be viewed as a set of interdependent queues. In general, the state-dependence in this equivalent network precludes a product-form solution, and the size of its state space grows rapidly with the number of classes and of jobs per class. This paper presents two methods for approximate solution of the equivalent state-dependent queueing network. The first approach is a manifold application of equivalence and decomposition. The second approach, less accurate than the first one, is a fast-converging iteration whose computational complexity grows near-linearly with the number of job classes and jobs in a class. Numerical examples illustrate the accuracy of the two methods.

Keywords and Phrases: queueing network models, simultaneous resource possession, multiprogramming, equivalence and decomposition, approximate solutions.

1.0 INTRODUCTION

There has been recently an increased interest in queueing network models with simultaneous resource possession in general (e.g. [1]), and, in particular, in models of computer systems with multiprogramming constraints, such as the one shown in Figure 1, where memory and other resources may be simultaneously held by a job [2]. When the performance measures to be obtained from the models are limited to throughputs and averages, fast and efficient iterative approaches exist (see, e.g. [3,4]). These approaches are close in spirit to the mean value analysis [5], and are not sufficient if one is also interested in distributions of, say, numbers of jobs in memory. The equivalence and decomposition method [6] seems a good candidate in such cases.

We restrict our attention to memory admission policies with a maximum multiprogramming degree (i.e., number of jobs in memory) per class of jobs. For such systems, a useful equivalence is the one depicted in Figure 2. There is one server per job class, and its service rate depends, in general, on the current numbers of jobs of each class in memory. These service rates can usually be obtained by decomposition, i.e., by analyzing a submodel of the multiprogramming set. Here, we assume that the service rates are given, and we concentrate on the solution of the equivalent network. As pointed out in [2], generally, such a network does not possess a product-form solution [7], and the size of its state space increases rapidly with the number od classes and the maximum numbers of jobs in each class. This renders the solution of the equivalent network a non-trivial task.

Two approximate solution methods for the state-dependent queueing network of Figure 2 are considered in this paper. The first approach, discussed in Section 2, is a manifold application of equivalence and decomposition. The second approach, presented in Section 3, is a fast-converging iterative method. It is less accurate than the first method but its computational complexity grows only linearly with the numbers of job classes and jobs in a class. Numerical examples illustrate the accuracy of the two approaches.

2.0 MANIFOLD EQUIVALENCE AND DECOMPOSITON

Let us start with the simple case of only one class of jobs in the system, depicted in Figure 3. Although this network can be solved directly for the total number of jobs in the system, it is generally advantageous to consider separately the distributions of the number of jobs in memory, and of the number awaiting admission, which we denote by $p(m)$ and $p(k)$, respectively.

For the number in memory, the equivalent Markovian queue, shown in Figure 4, has a finite capacity of M, the maximum degree of multiprogramming. The arrival rate is $(N-m)\lambda$, where N is the number of terminals, and $1/\lambda$ is the average think time of a user. The service rate is $v(m)$ for $m<M$, and $v(M)r$ for $m=M$, where $v(m)$ is the service rate in the network of Figure 3, and r is the conditional probability that the admission queue is empty given that there are M jobs in memory, i.e.,

$$r = \text{Prob}\{\ k=0 \mid M\ \} \quad .$$

The computation of $p(m)$ is straightforward [8], provided r is known. The latter is easily determined, since the conditional probability $\text{Prob}\{k \mid M\}$ may be obtained as the distribution of the number of users, k, in the single-server system of Figure 5. The arrival rate is $(N-M-k)\lambda$, and the constant service rate is $v(M)$. $p(k)$ is then simply

$$p(k) = p(M)\text{Prob}\{k \mid M\} + \sum_{m=1}^{M-1} p(m), \quad k=0,$$

$$p(k) = p(M)\text{Prob}\{k \mid M\} \ , \quad \text{for } k>0 \quad .$$

The above approach is easily shown to be exact, and, by separating the two distributions $p(k)$ and $p(m)$, it alleviates in many cases under/overflow problems that may appear in a direct solution for the total number of jobs in the system (see Appendix).

With multiple job classes, an approximate solution can be obtained by generalizing the two-class treatment of [9]. For simplicity, we use an example with three classes of jobs. Let $p(m_1,m_2,m_3)$ be the joint distribution for the numbers of jobs of each class in memory. The basis for our approach is the following identity

$$p(m_1,m_2,m_3) = p(m_3|m_2,m_1)p(m_2|m_1)p(m_1) \quad .$$

The conditional probabilities in the righthand side of this identity are approximately evaluated by disregarding transitions which change the condition variable(s). In this way, only single-class models of the type of that of Figure 3 have to be solved to obtain approximations for $p(m_3|m_2,m_1)$, $p(m_2|m_1)$ and $p(m_1)$.

The solution proceeds as follows. For given m_1, m_2, the service rate for class 3 is $v_3(m_1,m_2,m_3)$, locally a function of m_3 only. We thus compute $p(m_3|m_2,m_1)$ and $p(k_3|m_2,m_1)$ from a sigle-class model as considered above. We also compute

$$w_i(m_1,m_2) = \sum_{m_3=0}^{M_3} p(m_3|m_2,m_1)v_i(m_1,m_2,m_3) \ ,$$
$$i = 1,2,3 \ .$$

For given m_1, the service rate for class 2 is then $w_2(m_1,m_2)$, locally a function of m_2 only. The single-class model yields $p(m_2|m_1)$ and $p(k_2|m_1)$. We also compute

$$u_i(m_1) = \sum_{m_2=0}^{M_2} p(m_2|m_1)w_i(m_1,m_2) \ , \quad i = 1,2,3 \ ;$$

and

$$p(m_3|m_1) = \sum_{m_2=0}^{M_2} p(m_2|m_1)p(m_3|m_2,m_1) \ ;$$

$$p(k_3|m_1) = \sum_{m_2=0}^{M_2} p(m_2|m_1)p(k_3|m_2,m_1) \ .$$

Finally, $u_1(m_1)$ is the equivalent service rate for class 1. Hence we readily get $p(m_1)$ and $p(k_1)$. We also have

$$\theta_i = \sum_{m_1=0}^{M_1} p(m_1)u_i(m_1) \ , \quad i = 1,2,3 \ ;$$

$$p(m_j) = \sum_{m_1=0}^{M_1} p(m_1)p(m_j|m_1) \quad ;$$

and

$$p(k_j) = \sum_{m_1=0}^{M_1} p(m_1)p(k_j|m_1) \ , \quad \text{for } j>1 \ ,$$

where θ_i is the throughput for class i.

The above approach transforms the solution of the network of Figure 2 with c classes into c-fold solution of the single-class system of Figure 3. The only approximations introduced are in the computation of conditional probabilities; all other transformations are exact. With an appropriate choice of the order in which states are considered, it is possible to make the storage required for the approach grow linearly with the number of job classes and jobs in a class. The execution time, however, increases at the pace of growth of the size of the state space in the network of Figure 2. This is because in the manifold equivalence and decomposition described, one actually enumerates all the states of that network.

The next section is devoted to a faster, albeit generally less accurate, approach whose computational complexity grows near-linearly with the number of classes and the numbers of jobs per class.

3.0 ITERATIVE APPROACH

Our second approach to the solution of the state-dependent queueing network of Figure 2 is inspired from the iterative approach of [3]. The premise for the method is the assumption that, for any given job class, the influence of other classes is adequately represented by their average multiprogramming degrees. Hence, for simplicity, the service rate for class i jobs is approximated by $v_i(m_1^*,\ldots,m_i,\ldots,m_c^*)$, where m_j^* is the closest integer to $\bar{m}_j$, the average number of class j users in memory. For each class this results in the solution of the single-class system of Figure 3. Since the $\bar{m}_j$ (and hence, m_j^*) are known when the corresponding $p(m_j)$ have been computed, an iterative scheme suggests itself.

1. Choose initial values for m_j^*, $m_j^* \in [0,M_j]$, for $j=2,\ldots,c$.

2. For job classes $i=1,\ldots,c$, solve single-class model, and let m_i^* be the closest integer to $\bar{m}_i = \sum_{m_i} p(m_i)m_i$.

3. If maximum difference between old and new values for m_i^* is less than, say, 10^{-2}, exit; otherwise repeat step 2.

It has been the author's experience that this scheme converges within a small number of iterations, without much sensitivity to the choice of the starting values.

There are two potential sources of inaccuracies in the above approach. The first one is in the very premise of the method, which replaces the average of a function of a random variable with the function of the average of that variable. With strong non-linearities, this can result in large errors. The second source of inaccuracies is our use of the closest integer to avoid interpolation. This can induce errors even for linear functions if they are rapidly varying. In practice, however, the accuracy of the method is fair, as illustrated in the next section.

4.0 NUMERICAL RESULTS

In this section we present a few numerical results to illustrate the accuracy of the two approaches discussed in the preceding sections. Results of discrete-event simulations are used as reference values. Each simulation point corresponds to a total of 200,000 job completions. CPU utilizations by job classes are reported in lieu of throughputs. For the iterative approach, the number of iterations is given in parentheses.

Example 1: 2 classes of jobs

method	i	$\bar{k}_i$	$\bar{m}_i$	CPU utilization
simulation	1	0.789	6.478	0.638
	2	11.671	11.995	0.325
equivalence & decompositon	1	0.826	6.491	0.640
	2	11.633	11.995	0.327
iteration (3)	1	0.827	6.493	0.640
	2	11.570	11.995	0.329

The accuracy of the iterative method in this example seems representative of most cases. The accuracy of this approach in our second example, with a heavily loaded system, is among the worst observed.

Example 2: 3 classes of jobs

method	i	$\bar{k}_i$	$\bar{m}_i$	CPU utilization
simulation	1	27.114	9.991	0.040
	2	3.461	9.270	0.932
	3	25.494	10.987	0.027
equivalence & decomposition	1	27.559	9.947	0.037
	2	3.570	9.325	0.936
	3	25.675	11.000	0.027
iteration (2)	1	29.841	10.000	0.002
	2	3.570	9.325	0.936
	3	37.693	11.000	0.003

In this example, the iterative method underestimates the CPU utilizations for classes 1 and 3 by an order of magnitude. It may be noted, however, that the system is severely overloaded for these job classes, with an extremely poor response time. This is correctly identified by the method, even though the numerical values for the average response times would be considerably overestimated.

It is interesting to note that, under some load conditions, the manifold equivalence and decomposition method can be sensitive to class numbering, i.e., to the order in which classes are considered in the state reduction process.

Example 3: 3 job classes

method	i	$\bar{k}_i$	$\bar{m}_i$	CPU utilization
simulation	1	1.959	3.072	0.671
	2	7.355	1.944	0.061
	3	0.009	0.654	0.146
equivalence & decomposition order 1	1	2.032	3.086	0.673
	2	7.558	1.946	0.061
	3	0.009	0.658	0.147

equivalence & decomposition order 2	1	3.768	3.055	0.648
	2	7.681	1.946	0.061
	3	0.009	0.658	0.147
iteration (3)	1	3.463	3.419	0.647
	2	7.612	1.951	0.061
	3	0.009	0.658	0.147

Looking at the results of the equivalence and decomposition method we observe that, for class 1, a 4% relative difference in CPU utilizations is accompanied by a 33% relative difference in the total numbers of users in the system. Upon closer inspection, it turns out that this is intrinsic to the system studied itself, rather than a property of the solution method. Indeed, from the operational identity [10] $(N-\bar{n})\lambda=\theta$, where $\bar{n}=\bar{k}+\bar{m}$, and θ is the job throughput, we readily obtain

$$\frac{\Delta\bar{n}}{\bar{n}} \simeq -\frac{1}{N\lambda/\theta - 1}\frac{\Delta\theta}{\theta}$$

Thus, any relative difference in throughputs will get "amplified" by a factor $1/(N\lambda/\theta-1)$. As $N\lambda/\theta$ gets close to unity, the "amplification" factor grows. In our example, for class 1, we have $N=50$, $\lambda=3\cdot10^{-5}$, $\theta\simeq0.673/500$, which yields a difference amplification factor of 8.74. From Little's formula [8], we further obtain for the average response time

$$\frac{\Delta W}{W} \simeq \frac{N}{N-\bar{n}}\frac{\Delta\bar{n}}{\bar{n}} .$$

It is easily seen that, as system load increases and $\bar{n}$ moves closer to N, any relative difference in $\bar{n}$ will be amplified by a large factor. Thus small relative differences in throughputs can result in huge differences in average response times. Since the above "amplification" formulas are derived from operational identities, we conclude that the system studied in Example 3 operates in an unstable performance region for jobs of class 1.

Our next example illustrates the accuracy of the methods with 4 job classes and 2 CPUs.

Example 4: 4 job classes

method	i	$\bar{k}_i$	$\bar{m}_i$	CPU utilization
simulation	1	0.013	2.582	0.585
	2	0.002	2.719	0.093
	3	0.000	1.611	0.573
	4	25.127	0.999	0.367
equivalence & decomposition	1	0.012	2.594	0.587
	2	0.003	2.783	0.094
	3	0.000	1.621	0.576
	4	25.308	1.000	0.369
iteration (3)	1	0.012	2.594	0.587
	2	0.003	2.779	0.094
	3	0.000	1.622	0.576
	4	25.308	1.000	0.369

The following table gives the range for the relative difference in average response times between simulation and the two approximation methods for the set of examples presented. These values have been obtained using the "amplification" formula for response times.

Relative differences in average response times

example	1	2	3	4
equivalence & decomposition	.5% - 1%	2% - 8%	1% - 40%	.5% - 6%
iteration	1% - 1.5%	6% - 130%	1% - 40%	.5% - 6%

Finally, as an example, we show the distribution for the number of jobs in memory, $p(m_1)$, for class 1 of Example 1. The maximum multiprogramming degree for that class is 10.

Example 5: distribution of number in memory

	p(0)	p(1)	p(2)	p(3)	p(4)	p(5)	p(6)	p(7)	p(8)	p(9)	p(10)
simulation	.022	.047	.065	.077	.084	.088	.089	.086	.081	.072	.290
equivalence & decomposition	.022	.048	.065	.077	.084	.088	.089	.086	.080	.072	.288
iteration (2)	.022	.048	.065	.077	.084	.088	.089	.086	.080	.072	.288

As a whole, the accuracy of the manifold equivalence nad decomposition approach is good. The iterative approach is less accurate, but still produces fair results.

5.0 CONCLUSION

We have presented two methods for analyzing state-dependent networks that arise in the equivalence and decompostion approach to queueing network models of computer systems with multiprogramming constraints and multiple job classes. The first approach is a manifold application of equivalence and decomposition. It transforms the solution of a system with c classes into a c-fold solution of a single-class model. The storage requirements of the method are modest, but its execution time grows at the pace of the number of states in the state-dependent network being solved. The accuracy of its results is good. The second approach is a fast-converging iteration. It is generally less accurate than the first method, but its computational complexity grows near-linearly with the number of classes and jobs per class in the system. These methods have been presented for models with only interactive users. They can be extended to other types of job classes.

6.0 REFERENCES

1. P. Jacobson and E. W. Lazowska, The Method of Surrogate Delays: Simultaneous Resource Possession in Analytic Models of Computer Systems, Performance Evaluation Review 10 (1981) 165-174.

2. C. Sauer, Approximate Solution of Queueing Networks with Simultaneous Resource Possession, IBM J. Res. Develop. 25 (1981) 894-903.

3. Y. Bard, The Modeling of Some Scheduling Strategies for an Interactive Computer System, in: Computer Performance, K. M. Chandy and M. Reiser (editors), 1977, North-Holland Publ. Co., 113-137.

4. Y. Bard, A Simple Approach to System Modeling, Performance Evaluation 1 (1981) 225-248.

5. M. Reiser and S. S. Lavenberg, Mean Value Analysis of Closed Multichain Queueing Networks, J. ACM 27 (1980) 313-322.

6. A. Brandwajn, A Model of a Time-Sharing System Solved Using Equivalence and Decomposition Methods, Acta Inf. 4 (1974) 11-47.

7. F. Baskett, K. M. Chandy, R. R. Muntz and F. Palacios, Open, Closed, and Mixed Networks of Queues with Different Classes of Customers, J. ACM 22 (1975) 248-260.

8. L. Kleinrock, Queueing Systems, Vol.1: Theory (John Wiley & Sons, New York, 1975).

9. A. Brandwajn, A Model of a Time-Sharing System with Two Classes of Processes, in: Lecture Notes in Computer Science 34, GI-5 Jahrestagung, Dortmund, October 1975, Springer Verlag, Berlin 1975, 547-566.

10. P. J. Denning and J. P. Buzen, The Operational Analysis of Queueing Network Models, Computing Surveys 10 (1978) 225-241.

11. R. M. Brown, J. C. Browne, and K. M. Chandy, Memory Management and Response Time, Comm. ACM 20 (1977) 153-165.

12. R. M. Bryant, Maximum Processing Rates of Memory Bound Systems, J. ACM 29 (1982) 461-477.

APPENDIX

Denote by n the total number of jobs in the system, i.e., n=k+m. The probability distribution for n, p(n), is given by (see [6])

$$p(n) = \frac{1}{G} \prod_{i=1}^{n} (N-i+1)\lambda/v(i), \quad n = 1,\dots,M ;$$

$$p(n) = \frac{1}{G}\, p(M) \prod_{j=1}^{n-M} (N-M-j+1)\lambda/v(M), \quad n>M ,$$

where G is a normalization constant. Hence, the probability Prob{k | M} may be expressed as Prob{n=k+M}/Prob{n>=M}, i.e.,

$$\text{Prob}\{k \mid M\} = \frac{1}{H} \prod_{j=1}^{k} (N-M-j+1)\lambda/v(M) ,$$

where

$$H = \sum_{k=0}^{N-M} \prod_{j=1}^{k} (N-M-j+1)\lambda/v(M) .$$

This is identical to the distribution obtained from the analysis of the system of Figure 5. Hence, the conditional probability r, and, consequently, the distribution p(m) are also exact.

The approach presented in Section 2 separates the two distributions p(k) and p(m), effectively partitioning the values of n. Since, in the calculation of each p(k) and p(m) with their respective normalizing constants, there are fewer values to be considered than in the direct calculation of p(n), the possibility of under/overflow occuring is reduced.

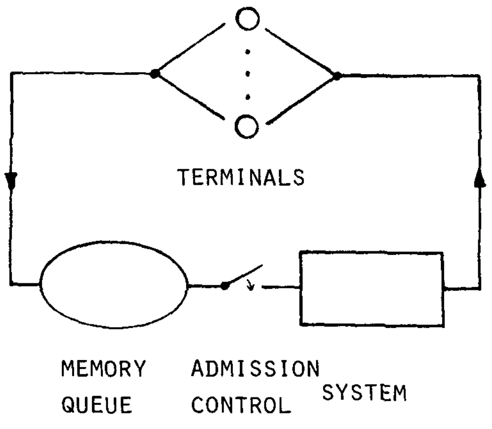


Figure 1: Model of an interactive system

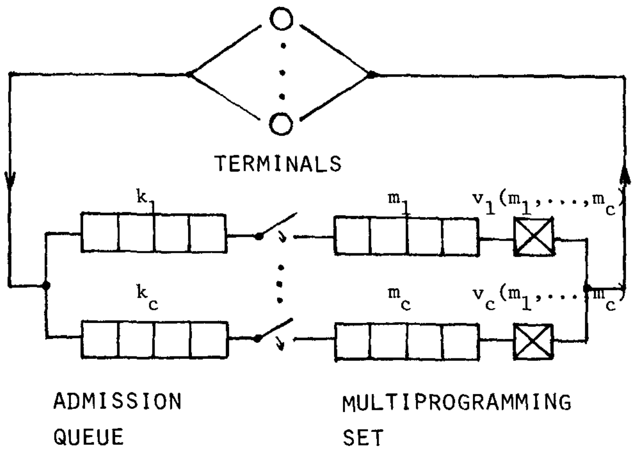


Figure 2: Equivalent queueing network

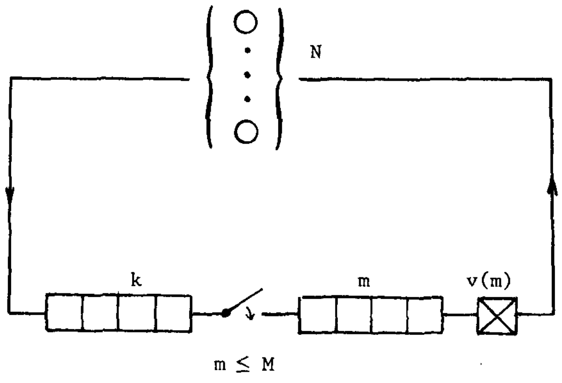
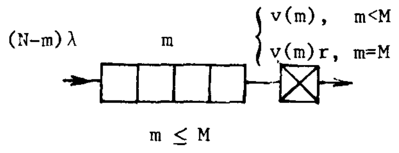


Figure 3: Equivalent network for a single-class system



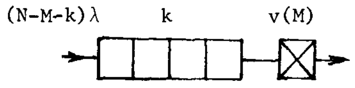


Figure 4: Equivalent system for p(m)

Figure 5: System for Prob{k|M}