Multiple Class Memory Constrained Queueing Networks

Edward D. Lazowska and John Zahorjan

Department of Computer Science
University of Washington
Seattle, WA 98195

ABSTRACT

Most computer systems have a memory constraint: a limit on the number of requests that can actively compete for processing resources, imposed by finite memory resources. This characteristic violates the conditions required for queueing network performance models to be *separable*, i.e., amenable to efficient analysis by standard algorithms. Useful algorithms for analyzing models of memory constrained systems have been devised only for models with a single customer class.

In this paper we consider the multiple class case. We introduce and evaluate an algorithm for analyzing multiple class queueing networks in which the classes have independent memory constraints. We extend this algorithm to situations in which several classes share a memory constraint. We sketch a generalization to situations in which a subsystem within an overall system model has a population constraint.

Our algorithm is compatible with the extremely time- and space-efficient iterative approximate solution techniques for separable queueing networks. This level of efficiency is mandatory for modelling large systems.

CR Categories and Subject Descriptors: C.4 [Performance of Systems]: *Modeling techniques;* D.4.8 [Operating Systems]: Performance -- *Modeling and prediction.*

General Terms: Performance

Additional Key Words and Phrases: computer system performance evaluation, queueing network model, approximate solution technique, memory constraint, population constraint.

1. Introduction

Queueing network models, in particular *separable* queueing network models [Baskett et al. 1975], are important tools in the design and analysis of computer systems. This is the case because, for many applications, separable queueing networks strike an appropriate compromise between accuracy and efficiency. Predictions accurate to within 5 to 10 percent for utilizations and throughput rates and to within 25 to 50 percent for response times are typical from these models. The existence of efficient solution techniques means that these predictions can be obtained in a matter of seconds. As a result, a large number of design alternatives may be investigated in a short period of time.

Although the class of separable queueing networks is fairly rich, certain structural characteristics of computer systems are difficult to represent within it. The existence of a *memory constraint* is one of these. Most systems have a memory constraint: a limit on the number of requests that can actively compete for processing resources, imposed by finite memory resources. In such a system, an arriving request for which memory is unavailable will be queued pending availability of memory. As an example, IBM's MVS operating system associates one or more "performance groups" (job classes) with each of several "domains" for which "target multiprogramming levels" are specified. A queueing network performance model of such a system may need to represent the effect of the memory constraint in order to achieve a useful level of accuracy. The alternatives for representing memory constraints within the class of separable queueing networks are limited:

- The population of a class of customers may be fixed. This accurately models a situation in which the system is always operating at memory saturation: there is a large external backlog of requests, and upon completion a request is immediately replaced in the system.
- The population of a class of customers may be unconstrained. This accurately models a situation in which the system never reaches memory saturation:

Lazowska's research is supported in part by the National Science Foundation under Grant No. MCS-8003344. Zahorjan's research is supported in part by the National Science Foundation under Grant No. MCS-8104879.

tion: there is always sufficient memory to accomodate an arriving request.

- Arrivals may be "lost" or "triggered" as a function of population [Lam 1977]. This accurately models a situation in which requests that arrive when memory is unavailable are discarded rather than queued.

Figure 1.1 illustrates an extremely common memory constraint phenomenon that violates the conditions required for separability. This computer system has a single customer class comprising M interactive users, at most C of which can simultaneously occupy memory. A request that arrives when there are already C active requests (C or more ready requests) is queued pending availability of memory. (In this example we ignore the details of swapping, and also assume that all requests have the same memory requirement.)

One alternative in analyzing this system is to ignore the memory constraint, yielding a separable queueing network model that can be analyzed efficiently. The resulting error may be unacceptably large. Another alternative is to simulate. The resulting cost may be unacceptably high. A third alternative, the most attractive in this case, is to use the well-known approximate analytic solution technique that we motivate and describe in the following paragraphs.

1.1. The Single Class Case

Separable queueing networks can include *load-dependent servers:* servers whose throughput rate varies as a function of their queue length. Chandy, Herzog and Woo [1975] showed that in single class separable networks, an exact solution results when an arbitrary subsystem is replaced by a single load-dependent server with throughput rates determined as follows:

- consider the subsystem in isolation by "shorting" its paths to the remainder of the queueing network;
- for each possible customer population, n, analyze the subsystem, obtaining its throughput rate with respect to the remainder of the queueing network, $\varphi(n)$;
- create a load-dependent server whose throughput rate with queue length n, $\mu(n)$, is equal to $\varphi(n)$.

Although it is perhaps surprising that results obtained in this manner are exact, the procedure is intuitively reasonable, since the queue length at the load-dependent server in the high-level model corresponds to the customer population of the subsystem in the original queueing network. Since the load-dependent server looks like the original subsystem to the remainder of the queueing network, it is called a *flow equivalent server.*

Nearly identical techniques can be used to define load-dependent servers that are "approximately" flow equivalent to subsystems in non-separable queueing networks. An approach that can be used to approximately analyze our example interactive system is described below and illustrated in Figure 1.2:

- consider the central subsystem in isolation by "shorting" its connection to the terminals;
- for each feasible customer population $n=1..C$, analyze the subsystem, obtaining its throughput rate with respect to the terminals, $\varphi(n)$;
- create a load-dependent server whose throughput rate with queue length n, $\mu(n)$, is defined by:

$$\mu(n) = \begin{cases} \varphi(n) & n=1..C \\ \varphi(C) & n>C \end{cases}$$

- solve a high-level model consisting of the full customer population, the terminals, and this load-dependent server.

Intuitively, this load-dependent server is flow equivalent to the original subsystem because the queue length at this server corresponds to the number of ready customers, and the throughput rate of the central subsystem is determined by the number of active customers, which is equal to the lesser of the memory capacity and the number of ready customers. The equivalence is approximate because the Chandy, Herzog and Woo theorem holds only for separable queueing networks. The approximation will yield excellent results because the terminals and the central subsystem are *nearly completely decomposable* in a formal sense [Courtois 1977].

Brandwajn [1974] first suggested this general approach to analyzing memory constrained systems. Keller [1976] extensively validated the technique in the form we have described it. This technique is successfully used with great regularity, and has been extended to single class models in which requests have distinct memory requirements [Brown et al. 1977, Bryant 1982]. Its utility arises both from its accuracy and from its efficiency. Its efficiency, in turn, depends on two factors:

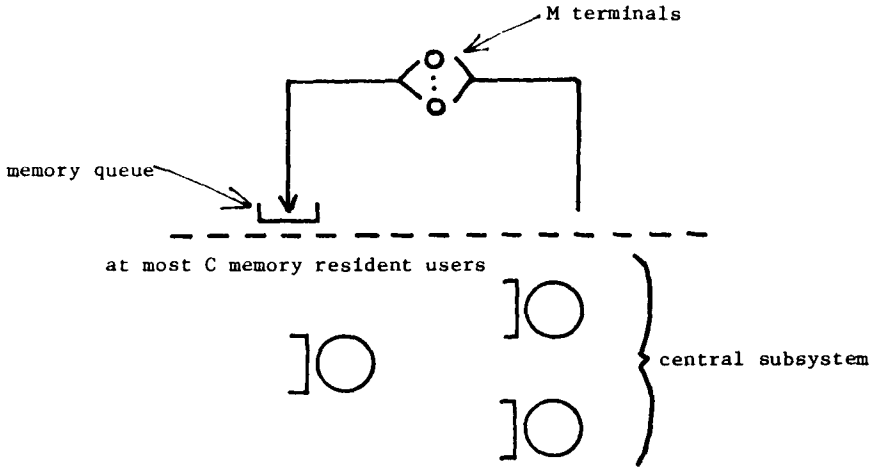

Figure 1.1 – A Simple Memory Constrained System

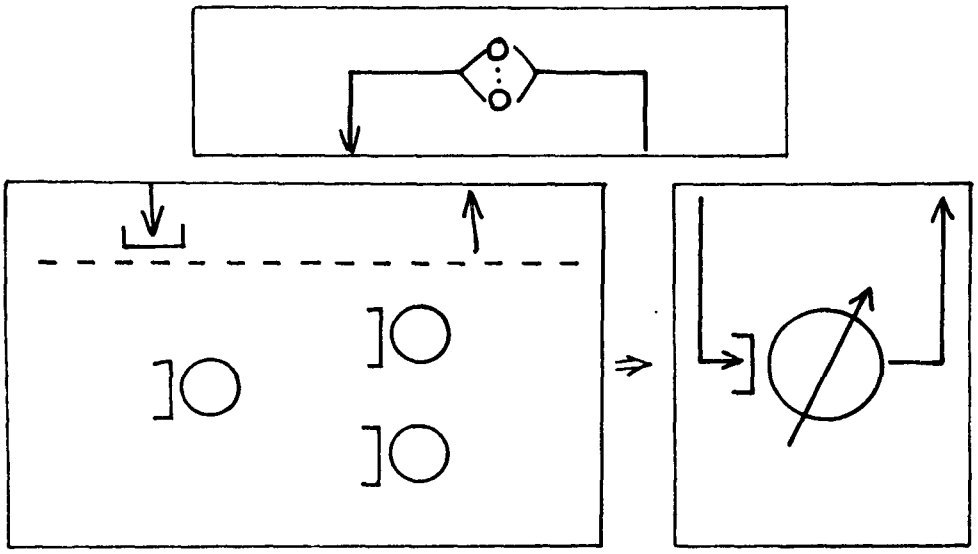

Figure 1.2 – An Approach Based On Approximate Flow Equivalence

- The load-dependent throughput rates used in defining the flow equivalent server can be obtained efficiently. In this case, the queueing network model of the central subsystem is a single class separable model. Exact analysis of this model requires time proportional to KC (K is the number of service centers and C is the maximum central subsystem population) and space proportional to C. The necessary rates can be obtained from a single analysis, since the computational algorithms calculate performance measures for populations from 1 to $n-1$ as a byproduct of calculating performance measures for population n.
- The resulting high-level queueing network model can be analyzed efficiently. In this case it is a single class separable model.

1.2. The Multiple Class Case

Now, consider a system with R customer classes, $r=1..R$, having independent memory constraints C_r. There is an obvious generalization of the above flow equivalence technique to this case:

- consider the central subsystem in isolation;
- analyze an R class separable queueing network for each feasible population vector $\vec{n}=(n_1,n_2,..,n_R)$ (a vector with an entry for each class indicating the population of that class), obtaining the throughput rate of each class r, $\varphi_r(\vec{n})$;
- use these rates to define a "population-vector-dependent" server with class-dependent throughput rates $\mu_r(\vec{n})$;
- solve a high-level model consisting of the full customer population, the terminals, and this server.

Recent experiments by Sauer [1981] have convincingly validated the accuracy of this generalized technique. Unfortunately, though, it violates both of the efficiency criteria set forth above:

- Although the requisite load-dependent throughput rates could be obtained by analyzing a single R class separable queueing network with population vector $\vec{n}=(C_1,C_2,..,C_R)$, the time and space required for this analysis each grow exponentially with the number of classes:

$$RK\prod_{r=1}^{R}(C_r+1)$$

  for time; less by a factor of R for space.
- The resulting high-level model is not separable, so can be solved exactly only by the extremely expensive global balance technique, the time and space requirements of which grow exponentially with the number of classes, customers, and service centers.

In this paper we present an algorithm for analyzing multiple class memory constrained queueing networks that, like the highly detailed approach evaluated by Sauer, is based on the concept of approximate flow equivalence. In designing our algorithm, we have consciously traded some of the accuracy of this highly detailed approach for a significant gain in efficiency: our algorithm runs in time proportional to KR^3 and in space proportional to $KR^2 + \max_r\{C_r\}$. Such a compromise is necessary if even moderately large problems are to be tractable; it is an important difference between our approach and the one recently proposed by Menasce and Almeida [1981].

In Section 2 we motivate and describe our basic technique, analyze its computational requirements, and evaluate it on the set of example queueing networks considered by Sauer. In Section 3 we extend our technique to situations in which several classes share a memory constraint, again evaluating it using data from Sauer's paper. In Section 4 we sketch a generalization to situations in which a subsystem within an overall system model has a population constraint. In Section 5 we summarize.

2. The Basic Technique: Independent Memory Constraints

In this section we present an algorithm for analyzing multiple class queueing networks with R customer

classes, $r=1..R$, which may be of three types: *interactive* (a fixed number of users cycling between terminals and the central subsystem), *transaction* (a stream of requests that arrive, obtain service, and depart), or *batch* (a fixed number of jobs). Classes of the interactive and transaction types may have independent memory constraints C_r.

Our algorithm was devised in the context of a larger effort to design a queueing network solution package suitable for use on very large problems. The core computational algorithms in this package, those used to analyze separable queueing networks, are the extremely time- and space-efficient iterative approximations based on Mean Value Analysis introduced by Bard [1979] and improved by Chandy and Neuse [1982]. These techniques can analyze an R class separable queueing network in space and in time per iteration proportional to KR^2 ; although the number of iterations required is somewhat sensitive to problem size, it is typically extremely small.

Our technique, like the detailed approach described in the previous section, is based on the concept of approximate flow equivalence. To achieve adequate efficiency we must avoid calculating load-dependent throughput rates $\varphi_r(\vec{n})$ for each of the feasible population vectors, because the number of such vectors grows exponentially with the number of classes. We must also ensure that the high-level model can be solved efficiently. To accomplish these two objectives we have made two *homogeneity assumptions:*

- that the throughput rate of class r with population n_r is dependent only on the mean populations of the other classes, conditioned on the class r population being n_r;
- that each class sees the other classes as though their central subsystem populations were independent of one another, obviating the above conditioning.

The former assumption accomplishes the first objective by allowing us to determine the load-dependent throughput rates of any class by analyzing an R class queueing network in which the populations of the other classes are fixed at their *average values*. These average values are determined from the high-level model; the high- and low-level models are solved iteratively, terminating when successive estimates are sufficiently close. The latter assumption accomplishes the second objective by allowing us to define a separate load-dependent server for each class. In essence, we analyze R separable single class high-level models, rather than a single non-separable R class high-level model.

2.1. The Algorithm

For ease in expressing the algorithm we denote the number of memory constrained classes by $\hat{R} \le R$ and order the classes so that the constrained classes have indices $r=1..\hat{R}$.

Algorithm 1 – Independent Memory Constraints

1. Obtain initial estimates of the average central subsystem customer populations for the memory constrained classes, $\bar{n}_r$ for $r=1..\hat{R}$. To do this, ignore all memory constraints, yielding a separable queueing network that can be efficiently analyzed. Set $\bar{n}_r$ to the minimum of C_r and the average class r central subsystem population observed in the unconstrained model.
2. In preparation for the iteration, change each of the $\hat{R}$ memory constrained classes into a batch class with population equal to $\bar{n}_r$. The non-integer customer populations that arise are naturally suited to the core computational algorithm.
3. For each memory constrained class $r=1..\hat{R}$:
   3.1. For each feasible population of the designated class, $n_r=1..C_r$, solve the queueing network obtaining the throughput rate of class r, $\varphi_r(n_r)$.
   3.2. Create a single class load-dependent server whose throughput rate with queue length n, $\mu_r(n)$, is defined by:
   $$\mu_r(n) = \begin{cases} \varphi_r(n) & n=1..C_r \\ \varphi_r(C_r) & n>C_r \end{cases}$$
   3.3. Solve a single class high-level model consisting of this load-dependent server and the "external environment" of class r (terminals or an arrival process). Obtain the queue length distribution at the load-dependent server, $p(n)$, and use this to calculate a new estimate for the average central subsystem population of class r:
   $$\bar{n}_r = \sum_{i=1}^{C_r} i\,p(i) \;+\; (1-\sum_{i=0}^{C_r} p(i))\,C_r$$
   Repeat Step 3 until successive estimates of the $\bar{n}_r$ for each class are sufficiently close.
4. Obtain performance measures for the memory constrained classes from the $\hat{R}$ high-level models solved during the final iteration. Obtain performance measures for the remaining classes by solving the queueing network defined in Step 2 using the final estimates for the $\bar{n}_r$.

2.2. Computational Requirements

Step 1 requires solving a single R class queueing network. Each execution of Step 3.1 requires solving C_r R class networks (the MVA-based approximations do not compute performance measures for sub-populations), each requiring time proportional to KR^2. Step 3.3 requires solving one single class network. A full execution of Step 3 requires looping $\hat{R}$ times. The number of iterations (full executions of Step 3) required is, in our experience, very small (typically less than 6), and is relatively insensitive to the size of the problem. Step 4 requires one additional solution of an R class network. The overall running time of our algorithm is thus proportional to KR^3:

$$\sum_{r=1}^{\hat{R}} C_r KR^2$$

Its space requirements are proportional to $KR^2 + \max_r \{C_r\}$.

2.3. Evaluation

Because useful analytic error bounds for approximate queueing network solution techniques are notoriously difficult to obtain, it is necessary to empirically evaluate such techniques. In this subsection we compare our approximation to simulations conducted by Sauer [1981] using IBM's Research Queueing Package (RESQ).

The basic queueing network simulated by Sauer is shown in Figure 2.1. It has a CPU, four equally-loaded disks, and two interactive customer classes with independent memory constraints. The number of users in each class, M_r, and the memory constraint for each class, C_r, differ in the various simulation runs. The remaining parameters are held constant, and are shown in Table 2.1.

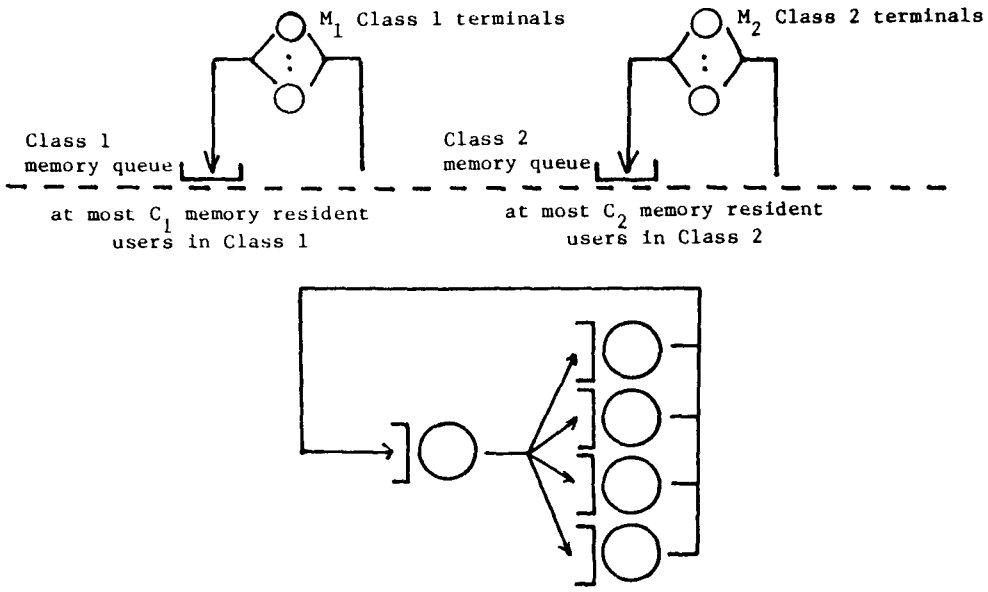

Figure 2.1 – The Example Queueing Network

parameter	Class 1	Class 2
average think time, seconds: (exponentially distributed)	5	10
average number of CPU-I/O cycles: (geometrically distributed)	10	20
average CPU service time per visit, msec.: (exponentially distributed, processor sharing discipline)	10	100
average disk service time per visit, msec.: (exponentially distributed, FCFS discipline, equal disk selection probability)	35	35

Table 2.1 – Parameters of the Example Queueing Network

Sauer chose three pairs of values for the populations of the two classes: (20,2), (30,3) and (40,4). For each of these pairs he experimentally selected three pairs of values for the memory constraints of the two classes: one providing low memory contention (i.e., little queueing for memory), one providing moderate memory contention, and one providing high memory contention. RESQ was used to simulate the system for each of these nine sets of parameter values. Each simulation was terminated when the relative width of the 90% confidence interval for class-independent mean response time (time from memory request to memory release) was 5%. Table 2.2 compares the results obtained by our approximate solution technique to the 90% confidence interval of the simulation for three performance measures: total CPU utilization, and average response time (in seconds) for Class 1 and for Class 2 users.

As Sauer notes, the simulation point estimates are not exact and the confidence intervals do not necessarily contain the corresponding true values. For these reasons it is difficult to judge the accuracy of the approximation. Given this caveat, the results are very encouraging:

- all but two of the approximate values for CPU utilization fall within the 90% confidence intervals; these two fall 1% and 4% outside;
- all but two of the approximate values for Class 1 response time fall within the 90% confidence intervals; these two each fall 1% outside;
- although half of the approximate values for Class 2 response time fall outside the 90% confidence intervals, the average distance is 3% and the worst case is 15%.

The greater errors for Class 2 are not surprising: the small population of this class stresses the accuracy both of our own algorithm (a small absolute error in the number of memory resident users will result in a large relative error in throughput rate) and of the underlying MVA-based approximations. Still, the accuracy of our technique is comparable to that of the highly detailed approach, at potentially great savings in space and time. In Section 3 we will discuss sources of error in greater detail.

An interesting aspect of Table 2.2 that deserves mention is the apparently anomolous behavior of Class 2, whose response time occasionally improves as the

M_1	M_2	C_1	C_2	CPU utilization		Cl. 1 response time		Cl. 2 response time	
				approx.	simul.	approx.	simul.	approx.	simul.
20	2	4	2	.61	(.60,.63)	.81	(.77,.80)	5.01	(4.49,5.09)
		3	1	.60	(.59,.61)	.93	(.90,.95)	5.01	(4.71,5.31)
		1	1	.49	(.48,.50)	4.89	(4.68,4.93)	3.80	(3.86,4.31)
30	3	7	2	.83	(.83,.85)	1.07	(1.03,1.08)	8.00	(6.70,7.64)
		5	1	.79	(.79,.81)	1.18	(1.13,1.19)	9.76	(8.42,9.69)
		2	1	.70	(.69,.71)	4.13	(3.97,4.17)	6.04	(6.08,6.95)
40	4	14	4	.92	(.96,.97)	1.56	(1.47,1.54)	16.2	(12.2,14.1)
		9	3	.95	(.95,.96)	1.71	(1.67,1.74)	12.9	(12.0,13.3)
		5	1	.86	(.87,.88)	2.35	(2.30,2.42)	15.5	(13.5,15.2)

Table 2.2 – Independent Memory Constraints

memory constraint becomes more severe. When such a change in the memory constraint occurs, queueing delay due to memory contention will increase, but queueing delay within the central subsystem will decrease. For each job class, the net effect may be either beneficial or deleterious. In the example, Class 2 is heavily CPU bound and the CPU is relatively heavily utilized. As the memory constraint becomes more severe, the increased time that Class 2 users spend queueing due to memory contention is sometimes more than offset by the decreased time that they spend queueing for the CPU once they become active.

Finally, a brief comment on execution times. Sauer's nine simulations required an average of 578 seconds of CPU time each on an IBM System/370 Model 168. Using Sauer's implementation of the detailed analytic approach, these examples required "less than 1/2 second each" on the same CPU. Our solutions required an average of 0.2 seconds each on a Digital VAX-11/780 without floating point accelerator, a machine roughly 25% the speed of a 168. More important than this comparison is the fact that these nine examples were relatively small, while the computational advantage of our technique can be expected to increase dramatically with problem size.

3. Shared Memory Constraints

In Section 2 we assumed that each class was subject to a memory constraint that was independent of the behavior of the other classes. In this section we extend our algorithm to shared memory constraints: constraints on the total number of jobs in memory, rather than on the populations of individual classes.

Let there be D *domains*, or shared regions of memory. Let C_d be the capacity of domain d, i.e., the number of jobs that can reside in that domain. Each memory constrained job class is assigned to a domain. Let $D(r)$ be a function that gives the domain number of class r if class r is memory constrained, and 0 otherwise. $M(d)$ is the inverse function, whose result is the set of classes belonging to domain d. To simplify the discussion we will assume that more than one class is assigned to each domain (i.e., that $D(r)=D(r')$ for some classes $r \neq r'$); dedicated domains are, of course, a special case of shared domains. We will consider both FCFS and priority (by class) scheduling for access to memory within domains.

Perhaps the simplest approach to this problem is a straightforward generalization of the algorithm in Section 2, with the only change being the calculation of the $\mu_r(n)$ in Step 3.2. This is replaced by

$$\mu_r(n) = \begin{cases} \varphi_r(n) & n \leq C_d - \delta_r \\ \varphi_r(C_d - \delta_r) & n > C_d - \delta_r \end{cases}$$

where $d \equiv D(r)$, and $\delta_r \equiv \sum_{s \in M(d), s \neq r} \bar{n}_s$. Thus, we view a domain shared by M classes as M smaller domains, each used by only a single class.

We have tried this simple approach on a set of examples related to those of the previous section, with mixed results. Because of the unreliability of this technique, we propose a slightly more complex algorithm for the shared memory constraint case. We develop this algorithm as the natural extension of a very general view of the problem, which we present next. Details of the algorithm follow this discussion.

3.1. A General Framework

We can view the solution of an R class memory constrained model as the solution of R distinct single class birth-death models. The behavior of each of these models can be visualized as shown in Figure 3.1. For each class r, the states of its model correspond to the number of class r customers competing for memory. Thus, for an open class the state space is infinite, while for a closed class the state space is finite, with individual states labeled from 0 to N_r, the number of customers in the class.

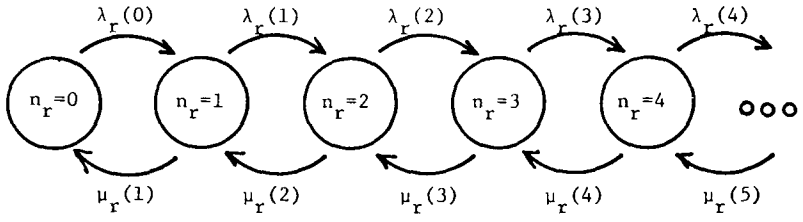

Figure 3.1 – Birth-Death Interpretation for an Open Class

The model changes from state n to states $n+1$ and $n-1$ with rates $\lambda_r(n)$ and $\mu_r(n)$, respectively. Intuitively, these parameters represent the rates at which class r customers request memory and at which class r customers complete service, respectively. We define a solution of the model to be the set of equilibrium state probabilities that satisfy the flow balance constraints: in equilibrium, the rate at which the network flows into any state must equal the rate at which it flows out of that state. In other words, the state probabilities for class r, $P_r(n)$, must satisfy the balance equations:

$$\lambda_r(n)\, P_r(n) = \mu_r(n+1)\, P_r(n+1), \quad \forall\, n$$
$$\sum_n P_r(n) = 1$$

The solution of models of this sort can easily be shown to be:

$$P_r(n) = P_r(0) \prod_{j=0}^{n-1} \frac{\lambda_r(j)}{\mu_r(j+1)}$$

Performance measures such as throughput and mean response time can be computed from the $P_r(n)$.

It is important to note that for any set of positive $\lambda_r(n)$ and $P_r(n)$ there exists a set of positive $\mu_r(n)$ such that the flow balance equations are satisfied. Thus, this view of the problem is always sufficient, in that it is possible to obtain exact solutions in all cases, if one could somehow deduce the $\lambda_r(n)$ and $\mu_r(n)$.

The computational efficiency of any solution technique based on this view of the problem depends upon the efficiency with which these state transition rates can be computed. Thus, we find it convenient to let the $\lambda_r(n)$ be the state dependent arrival rates to the system, since these are readily available from the input parameters of the model[1]. Given these values for the $\lambda_r(n)$, the $\mu_r(n)$ must be given by:

$$\mu_r(n) = \sum_{\vec{n}} \varphi_r(\vec{n} \mid n)\, p_r(\vec{n} \mid n)$$

where $\vec{n} \equiv (n_1, n_2, \ldots, n_R)$ is a vector representing the number of customers of each class currently in memory, $p_r(\vec{n} \mid n)$ is the probability that state $\vec{n}$ exists given that there are n customers of class r desiring memory, and $\varphi_r(\vec{n} \mid n)$ is the mean rate at which class r customers complete service when the state is $\vec{n}$ and there are n customers of class r competing for memory.

By classical decomposition theory [Courtois 1977], the original R class model is nearly completely decomposable into aggregates defined by those states corresponding to the same vector of customers competing for memory, since the rate at which customers leave the terminals is much smaller than the rate at which customers circulate among the service centers of the central subsystem. Thus, $\varphi_r(\vec{n} \mid n)$ differs only insignificantly from the unconditioned rate $\varphi_r(\vec{n})$. [2] This gives

$$\mu_r(n) \approx \sum_{\vec{n}} \varphi_r(\vec{n}) p_r(\vec{n} \mid n)$$
$$= \sum_{j=0}^{n} q_r(j \mid n) \sum_{\vec{n}\ s.t.\ n_r=j} \varphi_r(\vec{n}) p_r(\vec{n} \mid n,j) \qquad (3.1)$$

where $q_r(j \mid n)$ is the probability that j customers of class r are in memory given that n are competing, and $p_r(\vec{n} \mid n,j)$ is the probability of state $\vec{n}$ conditioned on n customers of class r competing for memory, j of which are resident.

[1] We note that because of the assumptions of the models with which we are dealing (specifically, Poisson external arrivals and exponentially distributed service times at terminal service centers), this interpretation of the $\lambda_r(n)$ is justified.

[2] This observation explains Sauer's success in applying the detailed decomposition technique directly to the R class model.

We note that any technique for calculating the $\mu_r(n)$ that requires an enumeration over all possible $\vec{n}$ is inherently unacceptable from a computational point of view, since the number of distinct $\vec{n}$ is of the order $\prod_{s=1}^{R}(N_s+1)$. A critical simplification in our approach, then, is to replace the inner summation in equation (3.1) with a single quantity representing class r's completion rate observed for a particular customer population. Define customer population $\vec{\eta}_{r,j}(j) \equiv (\bar{\eta}_1^r(j), \ldots, \bar{\eta}_{r-1}^r(j), j, \ldots, \bar{\eta}_R^r(j))$, where $\bar{\eta}_s^r(j)$ is the average number of class s customers in memory conditioned on there being j class r customers loaded. The inner summation of equation (3.1) is then replaced by $\varphi_r(\vec{\eta}_{r,j}(j))$, class r's completion rate with population $\vec{\eta}_{r,j}(j)$ in the subsystem.

This simplification would be exact if the completion rates of class r were linear in the number of customers of the other classes:

$$\varphi_r(\vec{n}) = \varphi_r(\vec{0}_j) \left[1 - \sum_{\substack{t=1 \\ t \neq r}}^{R} n_t \varepsilon_t\right] \qquad (3.2)$$

for some set of positive constants ε_t, where $\vec{0}_j$ denotes the state with j class r customers, and 0 customers of the other classes. This condition then gives:

$$\sum_{\vec{n}\ s.t.\ n_r=j} \varphi_r(\vec{n}) p_r(\vec{n} \mid n,j)$$
$$= \varphi_r(\vec{0}_j) \sum_{\vec{n}\ s.t.\ n_r=j} \left[1 - \sum_{\substack{t=1 \\ t \neq r}}^{R} n_t \varepsilon_t\right] p_r(\vec{n} \mid n,j)$$
$$= \varphi_r(\vec{0}_j) \left[1 - \sum_{\substack{t=1 \\ t \neq r}}^{R} \bar{\eta}_t^r(j) \varepsilon_t\right]$$
$$= \varphi_r(\vec{\eta}_{r,j}(j))$$

We have examined the behavior of a number of queueing networks to test the validity of the linearity assumption expressed by equation (3.2). We have observed that the completion rates of a designated class are surprisingly close to linear in the populations of other classes. Table 3.1 shows the completion rates of Class 2 as a function of the number of Class 1 customers for the central subsystem of the example used in Section 2. These rates are fairly linear over a wide range of values. It is important to also note that the $p_r(\vec{n} \mid n,j)$ typically have low variance, that is, they are significantly greater than zero only for a few "adjacent" $\vec{n}$. Thus, for the approximation to be reasonable in practice, the φ_r must be linear over only a small subspace of the $\vec{n}$, a much weaker condition.

		N_2		
		1	2	3
N_1	0	.370	.473	.495
	1	.305	.400	.429
	2	.258	.346	.378
	3	.223	.304	.336
	4	.196	.271	.303
	5	.174	.243	.275
	6	.156	.220	.251
	7	.141	.201	.231
	8	.128	.184	.231

Table 3.1 – Cl. 2 Completion Rates vs. Cl. 1 Population

The introduction of the above assumption results in a great reduction in the time and space requirements for calculating the $\mu_r(n)$, and leaves us with:

$$\mu_r(n) \approx \sum_{j=0}^{n} q_r(j \mid n)\varphi_r(\vec{n}_{r,j}(j)) \qquad (3.3)$$

Evaluating equation (3.3) requires only n operations, given the q_r and the φ_r.

3.2. The Algorithm

Specification of the algorithm requires that we first identify useful approximations for the quantities in equation (3.3). The calculation of the $\varphi_r(\vec{n}_{r,j}(j))$ is difficult because of the conditioning of $\vec{n}_{r,j}$ on j. We therefore introduce two homogeneity assumptions:

- that the $\vec{n}_{r,j}(j)$ are independent of j, giving:

  $$\varphi_r(\vec{n}_{r,j}(j)) = \varphi_r(\vec{n}_{r,j})$$

  where $\vec{n}_{r,j}$ is the population vector with the mean population observed by class r for all classes but r, and with j for r;
- that the mean number of customers in other classes seen by a class r customer is the equilibrium mean, which results finally in:

  $$\varphi_r(\vec{n}_{r,j}) = \varphi_r(\bar{N}_{r,j}) \qquad (3.4)$$

  where $\bar{N}_{r,j}$ is the population vector with the equilibrium mean population for all classes but r, and with j for r:

  $$\bar{N}_{r,j} = (\bar{n}_1, \bar{n}_2, \ldots, \bar{n}_{r-1}, j, \bar{n}_{r+1}, \ldots, \bar{n}_R)$$

While these assumptions are clearly not true in practice, we have found that the results obtained using them are quite good. Thus, we trade some potential increase in accuracy for computational efficiency.

The remaining problem is to approximate the $q_r(j \mid n)$ efficiently. Let $Q_r(j \mid n)$ be the probability that j customers from "competitor" classes are loaded into domain $d = D(r)$ given that there are n customers of class r present. A competitor class is any class in the same domain for FCFS scheduling, and any class in the same domain with equal or greater priority for priority scheduling. Then:

$$q_r(j \mid n) = \begin{cases} Q_r(C_d - j \mid n) & j < n \\ \sum_{k=0}^{C_d - j} Q_r(k \mid n) & j = n \end{cases} \qquad (3.5)$$

Our problem now is to approximate the $Q_r(j \mid n)$. To do so we make two assumptions:

- that $Q_r(j \mid n)$ is independent of n;
- that the Q_r behave as they would in a separable queueing network;

Using these assumptions, the Q_r can be calculated easily from information obtained during the solution of the R single class models.

The detailed statement of the algorithm, which follows, should serve to clarify matters. Throughout this description we will use d to denote $D(r)$, the domain to which class r belongs.

Algorithm 2 – Shared Memory Constraints

1. Let $\vec{L}_r = (L_r(1), L_r(2), \ldots, L_r(C_d))$ be a probability vector, where $L_r(j)$ represents an estimate for the probability that j customers of class r are loaded in memory at equilibrium. For all memory constrained classes r, initialize:

   $$L_r(j) = \begin{cases} 1 & j = 0 \\ 0 & \textit{otherwise} \end{cases}$$

2. Create a separable queueing network from the original memory constrained model by replacing all memory constrained classes with batch classes. The multiprogramming level of constrained class r is taken to be $\bar{n}_r = \sum_{j=1}^{C_d} j \cdot L_r(j)$, the current estimate for the mean class r multiprogramming level.
3. For each memory constrained class r :
   3.1. Solve the multiple class model of Step 2 with populations $\bar{N}_{r,j}$ for j varying over all feasible values from 1 to C_d.[3] Observe the system throughput rate $\varphi_r(\bar{N}_{r,j})$ of class r for each value of j.
   3.2. Compute $Q_r(j)$ as the jth component of the vector obtained by convolving together the $\vec{L}_s$ of all classes s that compete with class r for memory. For domains with FCFS scheduling, these are all other classes sharing the domain; for domains with priority scheduling, these are all other classes with equal or greater priority sharing the domain.
   3.3. Solve a single class high-level model consisting of class r's external environment and a single load-dependent service center. The service rates $\mu_r(n)$ of this center are computed using equations (3.3), (3.4), and (3.5). This solution yields a vector of probabilities $b_r(n)$ for the number n of class r customers competing for memory.
   3.4. Use the $b_r(n)$ to compute a new estimate for $\vec{L}_r$:

   $$L_r(j) = \sum_{n=j+1}^{\infty} b_r(n)\, Q_r(C_d - j) + b_r(j) \sum_{k=0}^{C_d - j} Q_r(k) \qquad (3.6)$$

   3.5. Calculate a new estimate for $\bar{n}_r$ from $\vec{L}_r$.

   Repeat Step 3 until successive estimates of the $\bar{n}_r$ for each class are sufficiently close.
4. Obtain performance measures for each memory constrained class from the most recent solution of the appropriate high-level model. Obtain performance measures for the unconstrained classes by solving the queueing network defined in Step 2 using the final estimates for the $\bar{n}_r$ of the constrained classes.

3.3. Computational Requirements

The space requirement of this algorithm is dominated by the space required to hold the $\vec{L}_r$ and to compute the solutions of the R class closed model. Thus, the space requirement is proportional to:

$$KR^2 + \sum_{s=1}^{R} C_{D(s)}$$

The time requirement is also dominated by these two steps, and thus is approximately:

$$R^2 \sum_{i=1}^{D} C_i^2 + R^3 \sum_{i=1}^{D} C_i$$

Note that these requirements are sufficiently small that queueing networks of any reasonable size may be solved. In constrast, any solution technique that requires the

[3] Note that it is a simple matter to model distinct memory requirements for each class by specifying domain capacities $C_{d,r}$ indicating the number of class r jobs that can be held in domain d.

exact solution of even a single R class separable network is severly limited in its applicability. Standard decomposition techniques are therefore not practicable in general.

3.4. Evaluation

As in Section 2, we present the results of applying our algorithm to a number of networks solved by Sauer using RESQ. The basic parameter settings of these models are the same as those given in Table 2.1. However, in all cases there is a single domain shared by both classes. Tables 3.2, 3.3 and 3.4 show the results of applying our technique to the model with FCFS, priority to Class 1, and priority to Class 2 memory scheduling, respectively. Within each table, we explore various numbers of users per class, and various domain capacities.

Although the tables (especially 3.3) indicate a few discrepancies, in general the accuracy of our technique is good. This is especially the case when one considers its small computational expense and the inaccuracy inherent in some of the assumptions that it makes.

For these examples, Sauer reports System/370 Model 168 CPU times of up to 1623 seconds using RESQ, and up to 28 seconds using the detailed analytic approach. Our technique required at most 5 seconds of VAX-11/780 CPU time; as in the case of independent memory constraints, the computational advantage of our technique can be expected to increase dramatically with problem size.

The existing error can be attributed to misestimates of the $\mu_r(n)$ due to various simplications used to increase efficiency. Some of these are:

- An approximation has been used to compute the $\varphi_r(\hat{N}_{r,j})$. The approximate technique is preferable to exact techniques because of the considerably greater computational expense of the latter. Additionally, the calculation of the $\varphi_r(\hat{N}_{r,j})$ in general requires the solution of a model with a non-integer number of customers in closed classes, since the average multiprogramming level is typically not an integer. If it were possible to obtain the $\varphi_r(\hat{N}_{r,j})$ more accurately without greatly increasing the computational expense of the algorithm, its accuracy would undoubtedly be improved somewhat.

M_1	M_2	C_1	CPU utilization		Cl. 1 response time		Cl. 2 response time	
			approx.	simul.	approx.	simul.	approx.	simul.
20	2	6	.62	(.60,.63)	.74	(.73,.76)	4.76	(4.34,4.93)
		4	.61	(.60,.62)	.85	(.91,.95)	4.73	(4.46,4.88)
		2	.58	(.53,.54)	1.55	(2.35,3.47)	4.38	(4.85,5.12)
30	3	9	.83	(.82,.84)	1.06	(1.00,1.04)	8.14	(6.95,7.92)
		6	.81	(.83,.85)	1.24	(1.20,1.26)	8.13	(6.87,7.47)
		3	.79	(.69,.71)	2.05	(3.44,3.62)	7.02	(6.69,7.01)
40	4	18	.94	(.95,.96)	1.52	(1.47,1.54)	14.33	(12.41,14.00)
		12	.93	(.95,.96)	1.61	(1.68,1.71)	14.85	(11.63,12.74)
		6	.89	(.90,.90)	2.16	(3.19,3.35)	13.89	(9.23,9.81)

Table 3.2 – Shared Memory Constraint, FCFS Memory Scheduling

M_1	M_2	C_1	CPU utilization		Cl. 1 response time		Cl. 2 response time	
			approx.	simul.	approx.	simul.	approx.	simul.
20	2	6	.62	(.60,.63)	.73	(.74,.77)	4.76	(4.39,5.02)
		4	.62	(.60,.62)	.76	(.89,.93)	4.72	(4.39,4.81)
		2	.60	(.49,.50)	1.28	(1.86,1.95)	4.37	(9.35,10.23)
30	3	9	.83	(.82,.84)	1.01	(1.04,1.09)	8.12	(6.78,7.61)
		6	.84	(.81,.82)	1.03	(1.29,1.35)	7.90	(7.27,7.92)
		3	.81	(.59,.60)	1.68	(2.32,2.43)	6.75	(24.07,27.30)
40	4	18	.94	(.95,.96)	1.51	(1.47,1.53)	14.42	(12.64,14.15)
		12	.95	(.95,.96)	1.42	(1.59,1.67)	14.70	(12.17,13.53)
		6	.97	(.81,.82)	1.56	(2.19,2.28)	12.45	(17.84,22.13)

Table 3.3 – Shared Memory Constraint, Priority to Class 1

M_1	M_2	C_1	CPU utilization		Cl. 1 response time		Cl. 2 response time	
			approx.	simul.	approx.	simul.	approx.	simul.
20	2	6	.62	(.62,.65)	.75	(.75,.78)	4.70	(4.56,5.11)
		4	.61	(.60,.62)	.92	(.91,.95)	4.55	(4.16,4.53)
		2	.55	(.54,.55)	2.78	(2.62,2.76)	3.80	(3.60,3.75)
30	3	9	.83	(.84,.86)	1.08	(1.06,1.11)	7.81	(7.24,8.11)
		6	.82	(.81,.82)	1.47	(1.41,1.48)	7.03	(6.22,6.67)
		3	.69	(.73,.74)	5.32	(4.19,4.40)	4.91	(4.74,4.98)
40	4	18	.94	(.95,.96)	1.52	(1.45,1.51)	14.19	(12.06,13.78)
		12	.95	(.95,.96)	1.69	(1.64,1.72)	12.95	(11.68,12.97)
		6	.88	(.91,.92)	4.22	(3.58,3.77)	8.11	(7.72,8.18)

Table 3.4 – Shared Memory Constraint, Priority to Class 2

- The computation of the $q_r(j \mid n)$ is in error, since we have assumed that the queue length distributions $\vec{l}_r$ behave as they would in a separable network, while the models we are considering are not separable. Unfortunately, the approximation is quite sensitive to errors in the $q(j \mid n)$. This sensitivity exists because the output rate of the central subsystem can be nearly linear in the number of customers resident in it, particularly if the subsystem is lightly used. In such cases, the expression for the approximate $\mu_r(n)$ given by equation (3.3) can be considerably in error, resulting in even larger errors in response times. This effect is most pronounced for models in which the central subsystem is lightly utilized, which occurs whenever the domain size is small. The ill effects of this can be seen easily by comparing the examples in this section: the smaller the domain size, the less accurate the result.
- $\vec{\eta}_{r,j}(j)$ is not independent of j. This simplification probably is the cause of much of the error in the examples. Unfortunately, it appears to be difficult to find a computationally feasible alternative that is more accurate.
- There are discrepancies in the way our algorithm (specifically, equation (3.5)) represents the details of domain scheduling. Specifically:
  - We do not, in fact, model FCFS scheduling, but rather "processor sharing".
  - We model preemptive priority scheduling, whereas Sauer simulates non-preemptive priority.

4. Subsystems With Population Constraints

Memory is not the only resource to impose a population constraint in computer systems. Rather, it is one specific instance of *simultaneous resource possession*, a general phenomenon that violates the separability conditions of queueing network models. In single class queueing networks, flow equivalent servers have been used to obtain approximate solutions for a number of simultaneous resource possession problems; for example, Chandy and Sauer [1978] use this approach to analyze a CDC batch system in which the number of "peripheral processors" (indistinguishable I/O controllers) places a limit on the number of disks that can be active simultaneously, and Jacobson and Lazowska [1982] use this approach to analyze a fairly general class of simultaneous resource possession problems.

The difference between modelling a memory constraint and modelling a more general population constraint arises in the high-level model: in the former case customers do not share resources outside of the population constrained subsystem (the "external environment" consists of terminals or an arrival process), while in the latter case they do (for example, jobs share the CPU when they are not contending for service at the population constrained I/O subsystem). Although the algorithm presented in Section 2 does not admit such sharing, it can be generalized to do so. The essence of this generalization is the replacement of the population constrained subsystem with R flow-equivalent servers, one for each class, whose load-dependent throughput rates are determined iteratively.

We require that the subsystem of interest impose an independent population constraint C_r on each of the classes $r=1..R$. The algorithm follows:

Algorithm 3 – Subsystems With Population Constraints

1. Obtain initial estimates of the average subsystem customer populations for all classes, $\bar{n}_r$ for $r=1..R$. To do this, ignore the population constraints, yielding a separable queueing network that can be efficiently analyzed. Set $\bar{n}_r$ to the minimum of C_r and the average class r subsystem population observed in the unconstrained model.
2. In preparation for the iteration, construct two queueing network models, a low-level model and a high-level model, each of which is easily analyzed:
   - The low-level model includes only those resources belonging to the population constrained subsystem. Each class r is represented as a batch class with population $\bar{n}_r$.
   - The high-level model includes those resources in the remainder of the system (the portion external to the population constrained subsystem) plus R load-dependent servers. Each class r visits its own load-dependent server, which represents the population constrained subsystem, plus appropriate other resources.
3. Iterate as follows:
   3.1. Consider the low-level model. For each class $r=1..R$:
   - For each feasible population of the designated class, $n_r=1..C_r$, solve this model obtaining the throughput rate of class r, $\varphi_r(n_r)$.
   - Create a single class load-dependent server whose throughput rate with queue length n, $\mu(n)$, is defined by:

$$\mu(n) = \begin{cases} \varphi_r(n) & n=1..C_r \\ \varphi_r(C_r) & n>C_r \end{cases}$$

   3.2. Consider the high-level model. Using the R load-dependent servers defined in Step 3.1, solve this model, obtaining the queue length distribution of each class r at its load-dependent server, $p_r(n)$. Use this distribution to calculate a new estimate for the average subsystem population of class r:

$$\bar{n}_r = \sum_{i=1}^{C_r} i\, p_r(i) \;+\; (1-\sum_{i=0}^{C_r} p_r(i))\,C_r$$

   Repeat Step 3 until successive estimates of the average subsystem population for each class are sufficiently close.
4. Obtain performance measures directly from the high-level model.

This algorithm has been programmed and used with good results. Although it has not been extensively evaluated, there is every reason to believe that its behavior will be comparable to that of the algorithm in Section 2. In the manner of Section 3, it can be extended to situations in which several classes share a population constraint. Classes without population constraints can be handled by imposing artificial constraints that are (almost) never reached; appropriate values can be determined from the queue length distributions at the appropriate load-dependent servers.

5. Summary

Separable queueing network models are important tools in the design and analysis of computer systems because, for many applications, they strike an appropriate compromise between accuracy and efficiency. Although the class of separable queueing networks is fairly rich, certain characteristics of computer systems that can have significant impact on system performance, such as simultaneous resource possession in general and memory constraints in particular, cannot be modelled by separable networks. Specialized solution techniques must be devised to represent the effects of these characteristics.

In this paper we have introduced a technique for analyzing multiple class queueing networks in which the classes have independent memory constraints. We have extended our technique to situations in which several classes share a memory constraint. We have sketched a generalization to situations in which a subsystem within an overall system model has a population constraint.

Our technique was devised within the context of a larger effort to design a queueing network solution package suitable for use on very large problems. Our algorithm is compatible with the extremely time- and space-efficient iterative approximate solution techniques for separable networks. This level of efficiency is mandatory; achieving it has cost very little in terms of overall accuracy.

Acknowledgements

Pat Jacobson suggested the generalization to population constrained subsystems described in Section 4. Charlie Sauer assisted us in interpreting his simulation results.

References

[Bard 1979]
Yonathan Bard; "Some Extensions to Multiclass Queueing Network Analysis"; Proc. IFIP W.G.7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation, Vienna, February 1979.

[Baskett et al. 1975]
Forest Baskett, K. Mani Chandy, Richard R. Muntz and Fernando G. Palacios; "Open, Closed and Mixed Networks of Queues with Different Classes of Customers"; *JACM 22*, 2, April 1975, pp. 248-260.

[Brandwajn 1974]
Alexandre Brandwajn; "A Model of a Time-Sharing System Solved Using Equivalence and Decomposition Methods"; *Acta Informatica 4*, 1, 1974, pp. 11-47.

[Brown et al. 1975]
R.M. Brown, J.C. Browne and K.M. Chandy; "Memory Management and Response Time"; *CACM 20*, 3, March 1977, pp. 153-165.

[Bryant 1982]
R.M. Bryant; "Maximum Processing Rates of Memory Bound Systems"; *JACM 29*, 2, April 1982, pp. 461-477.

[Chandy et al. 1975]
K.M. Chandy, U. Herzog and L. Woo; "Parametric Analysis of Queueing Networks"; *IBM J. Res. Develop. 19*, 1, January 1975, pp. 36-42.

[Chandy & Neuse 1982]
K.M. Chandy and D. Neuse; "Fast Accurate Heuristic Algorithms for Queueing Network Models of Computing Systems"; *CACM 25*, 2, February 1982.

[Chandy & Sauer 1978]
K. Mani Chandy and Charles H. Sauer; "Approximate Methods for Analyzing Queueing Network Models of Computing Systems"; *Computing Surveys 10*, 3, September 1978, pp. 281-317.

[Courtois 1977]
P.J. Courtois; *Decomposability: Queueing and Computer System Applications;* Academic Press, 1977.

[Jacobson & Lazowska 1982]
Patricia A. Jacobson and Edward D. Lazowska; "The Method of Surrogates: Simultaneous Resource Possession in Queueing Network Models of Computer Systems"; *CACM 25*, 2, February 1982.

[Keller 1976]
T.W. Keller; "Computer System Models with Passive Resources"; Ph.D. Thesis, University of Texas at Austin, 1976.

[Lam 1977]
S.S. Lam; "Queuing Networks with Population Size Constraints"; *IBM J. Res. Develop. 21*, 4, July 1977, pp. 370-378.

[Menasce & Almeida 1981]
Daniel A. Menasce and Virtilio A.F. Almeida; "Computing Performance Measures of Computer Systems with Variable Degree of Multiprogramming"; Proc. CMG XII, pp. 97-106.

[Sauer 1981]
Charles H. Sauer; "Approximate Solution of Queueing Networks with Simultaneous Resource Possession"; *IBM J. Res. Develop. 25*, 6, November 1981, pp. 894-903.