Charles H. Sauer

Approximate Solution of Queueing Networks with Simultaneous Resource Possession

Queueing networks are important as performance models of computer and communication systems because the performance of these systems is usually principally affected by contention for resources. Exact numerical solution of a queueing network is usually only feasible if the network has a product form solution in the sense of Jackson. An important network characteristic which apparently precludes a product form solution is simultaneous resource possession, e.g., *a job holds memory and processor simultaneously. This paper extends previous methods for approximate numerical solution of queueing networks with homogeneous jobs and simultaneous resource possession to networks with heterogeneous jobs and simultaneous resource possession.*

1. Introduction

A major objective of computing systems (including computer communication systems) development in the last two decades has been to promote sharing of system resources. Sharing of resources necessarily leads to contention, *i.e.*, queueing, for resources. Contention and queueing for resources are typically quite difficult to quantify when estimating system performance. A major research topic in computing systems performance in the last two decades has been solution and application of queueing models. These models are usually networks of queues because of the interactions of system resources. For general discussion of queueing network models of computing systems, see Sauer and Chandy [1] and recent special issues of *Computing Surveys* [2] and *Computer* [3].

Much of the attention in queueing network research has been given to models with a *product form solution* in the sense that

$$P(S_1, \cdots, S_M) = \frac{P_1(S_1) \cdots P_M(S_M)}{G},$$

where $P(S_1, \cdots, S_M)$ is the probability of a network state in a network with M queues, $P_m(S_m)$, $m = 1, \cdots, M$, is a factor corresponding to the probability of the state of queue m, and G is a normalizing constant. The existence of a product form solution for a model makes numerical solution feasible where a large number of queues and/or jobs would otherwise make numerical solution infeasible. Since the original work of Jackson [4], it has been shown that the product form solution exists for networks with heterogeneous jobs, several important scheduling disciplines, and state-dependent behavior [5–7]. Efficient computational algorithms have been developed for these networks [8–11].

However, there are a number of system characteristics which apparently preclude a product form solution. Among the most important of these is simultaneous resource possession, *i.e.*, a job's activities require simultaneous possession of more than one resource, *e.g.*, memory and processor. If there is significant contention for only one of the simultaneously held resources, then the model may ignore the others. Otherwise, one must usually settle for an approximate numerical solution [12, 13] or simulation.

This paper focuses on approximate numerical solution of models with simultaneous resource possession in cases such as the one depicted in Fig. 1. In this case a job holding memory may simultaneously also hold the CPU or simultaneously also hold a disk. Further, requests for and releases of simultaneously held resources are nested, *i.e.*, memory is requested before the CPU is requested

and released after a disk is released. With such nesting one can transform the original network into one of the form of Fig. 2. A solution of the transformed network can then be interpreted as an approximate solution of the original network. The approximate solution will usually be much less expensive than simulation. It is quite difficult to estimate the error in the approximate solution, but empirical studies suggest the error is acceptable in many situations. Some of the influential works using this approach for models similar to this one are those of Brandwajn [14], Brown [15, 16], Courtois [17, 18], and Keller [19]. This general approach can be applied to other resources and to more than two simultaneously held resources. For an introduction to previous work using this approach, see the survey papers by Chandy and Sauer [12, 13]. Two other approaches to solution of this problem are those of Bard [20] and Jacobson and Lazowska [21].

Except for the work of Bard [20] and Newsom and Ward [22], previous efforts have assumed that jobs were homogeneous. Our interest here is extending the above general approach to networks with heterogeneous jobs. Though Bard's approach applies to fairly general cases with heterogeneous jobs, we find it less accurate than our approach for many cases; see the Appendix for further discussion. Newsom and Ward attempted to extend Brown's work [15, 16] to heterogeneous jobs. Though they were able to extend the relevant equations, the result was computationally impractical, as they discussed. Further, only two test cases were considered in their paper.

In Section 2 we consider the model of Fig. 1 *without* memory contention in order to develop previous results on *flow-equivalence*, the basis of this work. In Section 3 we consider the model of Fig. 1 with heterogeneous jobs where separate memory is dedicated to each type of job. In Section 4 we consider the case where the different types of jobs share memory. Empirical results demonstrate the effectiveness of the methods used.

2. Flow-equivalence

A principal objective in the transformation of the network of Fig. 1 to the network of Fig. 2 is to obtain *flow-equivalence*, *i.e.*, the flow of jobs through the composite queue of Fig. 2, given a specific population of jobs in that queue, is equivalent to the flow of jobs through the corresponding subnetwork of Fig. 1, given the same population of jobs in the subnetwork. This can be approached either by using "Norton's Theorem" for queueing networks, analogous to Norton's Theorem for electrical circuits [23], or by using concepts of weak-coupling of subnetworks [17]. We use the Norton's Theorem approach.

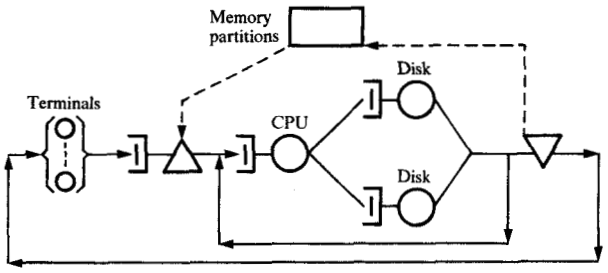

Figure 1 Queueing network model of interactive computer system.

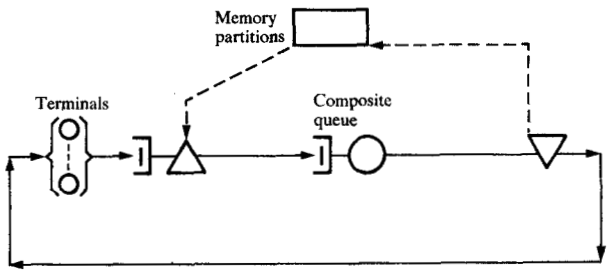

Figure 2 Computer system model with CPU and I/O disks replaced by composite queue.

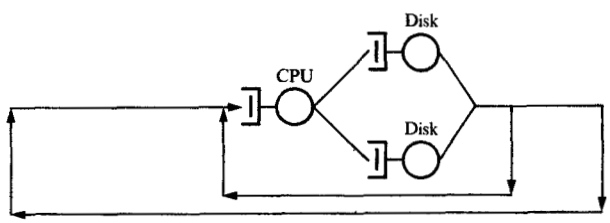

Figure 3 CPU and disk subnetwork.

Let us assume that there are two types of jobs, numbered 1 and 2. Extension to more than two types of jobs is conceptually trivial. (Extension to more than a few job types may be computationally prohibitive, as with other queueing network problems.) Let the number of type k jobs be N_k, $k = 1, 2$.

The Norton's Theorem approach to this problem is straightforward. We would consider the network of Fig. 1 with the terminals queue "shorted," *e.g.*, with service time set to zero. Our intermediate objective is to obtain the throughput through the "short." To obtain the desired throughput, we need only solve the network of Fig. 3 to obtain the throughput through the outer loop. Let

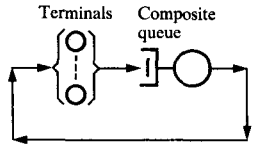

Figure 4 Computer system model with memory, CPU, and disk replaced by composite queue.

$R_k(n_1, n_2)$ be the throughput of type k jobs through the outer loop of Fig. 3 given n_1 type 1 jobs and n_2 type 2 jobs in that network, $k = 1, 2$, $n_1 = 0, \cdots, N_1$, $n_2 = 0, \cdots, N_2$. Then let the composite queue of Fig. 2 have service rate $\mu_k(n_1, n_2) = R_k(n_1, n_2)$ for type k jobs when there are n_1 type 1 jobs and n_2 type 2 jobs in the composite queue. Both types of jobs are in service simultaneously when both types of jobs are present in the queue.

Since we have assumed for the moment that there is no memory contention, the network of Fig. 2 is equivalent to the network of Fig. 4. The network of Fig. 4 satisfies product form *if the original network satisfies product form* (assuming no memory contention). Several computational algorithms for product form networks apply to the network of Fig. 4 if it satisfies product form [11]. However, composite queues with general functions of the form $\mu_k(n_1, n_2)$, such as the ones we obtain below, do not necessarily satisfy product form conditions.

Assuming that the network of Fig. 1 satisfies product form (assuming no memory contention), then the throughputs through the terminals are the same in the network of Fig. 1 and the networks with the composite queue (with nonzero service time at the terminals). Further, the performance measures for the CPU and disk queues can be obtained from the solutions of the networks of Fig. 2 and Fig. 3. Throughputs are immediately available by flow arguments. Marginal queue length distributions and moments of queue length can be obtained as weighted sums where the weights are the values of the composite queue marginal queue length distribution. For example, mean queue length of type k jobs at the CPU can be obtained as

$$L_{k,\mathrm{CPU}} = \sum_{n_1=0}^{N_1} \sum_{n_2=0}^{N_2} P(n_1, n_2) L^*_{k,\mathrm{CPU}}(n_1, n_2),$$

where $P(n_1, n_2)$ is the marginal probability of n_1 type 1 jobs and n_2 type 2 jobs in the composite queue and $L^*_{k,\mathrm{CPU}}(n_1, n_2)$ is the mean queue length of type k jobs at the CPU in the network of Fig. 3, given n_1 type 1 jobs and n_2 type 2 jobs in that network. Mean queueing times can be obtained by Little's Rule. Utilizations can be obtained either from throughputs or marginal queue length distributions, depending on the characteristics of the individual queues. For more discussion of individual queue measures see Sauer and Chandy [1, Section 6.3.3]. *This entire process is exact provided that the original network satisfies product form.*

3. Dedicated resources

In this section we consider the case that the outer resource in the nesting (memory in our example) is managed so that different types of jobs have different dedicated units of the resource. The inner resource in the nesting (CPU or disk in our example) is shared among all job types. This case is more tractable than the fully shared resource case of Section 4.

Let us assume that memory is organized in T partitions and that each job requires exactly one partition. In this section we assume that there are T_1 partitions dedicated to type 1 jobs and T_2 partitions dedicated to type 2 jobs ($T_1 + T_2 = T$). We assume in this section that, within a job type, memory is scheduled First-Come-First-Served (FCFS), independent of the other type. The discussion in this section extends directly to the more general memory organizations considered by Brown for homogeneous jobs [16].

Now we return to the case of interest, *i.e.*, where there is memory contention. Essentially the above process is followed, but there are two new issues to be considered. First, the solution of the network of Fig. 2 no longer gives exact values for the measures for the network of Fig. 1. This is easily demonstrated by example, but it is quite difficult to characterize the amount of error.

The second issue is the solution of the network of Fig. 2. We can immediately transform the network of Fig. 2 to that of Fig. 4 by recognizing that there will never be more than T_k type k jobs in the composite queue, $k = 1, 2$. Thus, we can let the service rate function for the composite queue be

$$\mu_k(n_1, n_2) = R_k(\min(T_1, n_1), \min(T_2, n_2)), \quad (1)$$

where $k = 1, 2$, $n_1 = 0, \cdots, N_1$, $n_2 = 0, \cdots, N_2$.

A composite queue with this rate function does not necessarily satisfy product form. However, the network of Fig. 4 with this rate function is easily solved by considering the underlying Markov process. Many numerical approaches apply readily to such a Markov process. Gauss-Seidel iteration and the related methods considered by Stewart [24] are obvious possibilities. Brandwajn's recent iterative method could be used [25].

We, somewhat arbitrarily, choose to apply Herzog's method [26] to this problem; see [27] for details.

Since it is difficult to characterize the error introduced by the replacement of the subnetwork of Fig. 1 by the composite queue, it is necessary to empirically evaluate the approximate method. Ideally, one would run experiments for a wide parameter space of models. In our experiments we fixed several parameters (see Table 1) and varied N_1, N_2, T_1, and T_2. Three pairs of values were used for (N_1, N_2), (20, 2), (30, 3), and (40, 4). T_1 and T_2 were chosen as follows. For a given (N_1, N_2) pair, the network was evaluated assuming no memory contention. (All of the networks evaluated satisfy product form except for memory contention.) One (T_1, T_2) pair was chosen so that T_k, $k = 1, 2$, was approximately equal to the mean memory queue length of type k jobs, $L_{k,\text{Memory}}$, in the network without memory contention. In other words, $T_k \simeq N_k - L_{k,\text{Terminals}}$, where $L_{k,\text{Terminals}}$ is the mean type k queue length at the terminals in the network without memory contention. This should result in moderate memory contention. The other two pairs were chosen to have T 50% larger, *i.e.*, minimal memory contention, and to have T 50% smaller, *i.e.*, severe memory contention. Table 2 gives the nine parameter combinations.

Reference values were obtained by simulation using the Research Queueing Package (RESQ) [28, 29]. Confidence intervals were obtained using the regenerative method [30]. A sequential stopping rule [31] was used to obtain a relative width of 5% for the confidence interval for the type-independent mean response time (mean time from memory request to memory release for all jobs) at a 90% confidence level. Table 2 also gives the CPU time in seconds spent on these simulations on an IBM 370/168. For each of the nine cases, the CPU time for approximate solution was less than one-half second on a 168.

Since the simulation point estimates are not exact, and a confidence interval does not necessarily contain the corresponding true value, it is not clear how accuracy of the approximation values should be judged. The following criteria are somewhat arbitrary and may be more stringent than required by many applications. Given a performance measure ν from the approximation, let us construct an interval $(\nu - \delta, \nu + \delta)$ such that it is the smallest interval that (partially) overlaps the simulation confidence interval for this measure. If $\delta \leq 0.05 \times \nu$, then the approximation value is considered satisfactory; if $\delta \leq 0.10 \times \nu$, then the approximation value is considered in error; and if $\delta > 0.10 \times \nu$, then the approximation value is considered severely in error.

These criteria were applied to the approximation results for all queues (terminals, memory, CPU, and disks) for both type-dependent and type-independent measures of utilization, throughput, mean queue length and mean queueing time. (The memory queueing time is defined as time from request until release.) The approximation results were satisfactory for all nine parameter combinations for all measures except for type-dependent CPU utilization. The type 1 CPU utilization was underestimated and in error for all nine combinations. The type 2 CPU utilization was overestimated and in error for all combinations except combination 3, where the type 2 CPU utilization was satisfactory. Table 3 gives the approximation values and simulation confidence intervals for type-independent CPU utilization (U) and mean response times (memory queueing time Q) and type-dependent mean response times. The achieved accuracy seems more than adequate for most applications, and the approximation solutions required roughly three orders of magnitude less computation than the simulations.

Table 1 Parameters fixed for all experiments.

Mean think time: type 1—5 seconds, type 2—10 seconds. (exponential distribution, "Infinite Server" discipline)
Mean number of CPU-I/O cycles: type 1—10, type 2—20. (geometric distribution)
Mean CPU service time: type 1—10 ms, type 2—100 ms. (exponential distribution, Processor Sharing discipline)
Four identical disks with same parameters for each type: Branching probabilities from CPU to disk: 0.25. Mean disk service time: 35 ms. (exponential distribution, FCFS discipline)

Table 2 Dedicated memory parameter combinations.

Case	N_1	N_2	T_1	T_2	*Simulation time* (s)
1	20	2	4	2	403
2	20	2	3	1	525
3	20	2	1	1	330
4	30	3	7	2	507
5	30	3	5	1	594
6	30	3	2	1	442
7	40	4	14	4	603
8	40	4	9	3	984
9	40	4	5	1	816

4. Shared resources

In this section we consider the case that the outer resource in the nesting (memory in our example) is managed so that different types of jobs share the same

Table 3 Dedicated memory performance measures.

Case	U_{CPU}		Q_{Memory}		$Q_{1,Memory}$		$Q_{2,Memory}$	
1	0.62	(0.60, 0.63)	0.95	(0.91, 0.95)	0.80	(0.77, 0.80)	4.66	(4.49, 5.09)
2	0.60	(0.59, 0.61)	1.08	(1.06, 1.11)	0.93	(0.90, 0.95)	5.08	(4.71, 5.31)
3	0.49	(0.48, 0.50)	4.79	(4.64, 4.87)	4.83	(4.68, 4.93)	4.09	(3.86, 4.31)
4	0.84	(0.83, 0.85)	1.28	(1.24, 1.30)	1.06	(1.03, 1.08)	7.59	(6.70, 7.64)
5	0.80	(0.79, 0.81)	1.42	(1.37, 1.44)	1.17	(1.13, 1.19)	9.13	(8.42, 9.69)
6	0.69	(0.69, 0.71)	4.23	(4.09, 4.30)	4.10	(3.97, 4.17)	6.44	(6.08, 6.95)
7	0.96	(0.96, 0.97)	1.83	(1.79, 1.87)	1.50	(1.47, 1.54)	13.30	(12.15, 14.10)
8	0.95	(0.95, 0.96)	2.00	(1.98, 2.07)	1.69	(1.67, 1.74)	12.46	(11.98, 13.26)
9	0.87	(0.87, 0.88)	2.69	(2.65, 2.78)	2.33	(2.30, 2.42)	14.47	(13.48, 15.15)

units of that resource. Thus, all types of jobs share exactly the same simultaneously held resources. This case is, perhaps, more important than the dedicated resource case. Unfortunately, the shared resource case also seems more difficult. The difficulty is in transforming the network of Fig. 2 to that of Fig. 4, to avoid the relatively expensive solution of the network of Fig. 2. The difficulty of this transformation depends on the memory scheduling discipline; for some of the most interesting disciplines an accurate transformation does not seem feasible.

For strictly preemptive priority, the appropriate transformation seems, assuming type 1 jobs have higher priority, to use

$$\mu_k(n_1, n_2) = R_k(\min(T, n_1), \min(T - \min(T, n_1), n_2)), \quad (2)$$

$k = 1, 2$. Corresponding to Eq. (1), Eq. (2) gives a simple and intuitively reasonable basis for selecting a specific $R_k(i_1, i_2)$ to use for $\mu_k(n_1, n_2)$ when $n_1 + n_2 > T$.

For other disciplines, such as FCFS or non-preemptive priority, there seems to be no comparably simple or intuitively reasonable basis for the transformation. The problem is that the representation of memory and the nested subnetwork by a function of the form $\mu_k(n_1, n_2)$ discards essential state information. For the FCFS discipline we investigated a number of weighted sum approaches, *e.g.*, equal weights:

$$\mu_k(n_1, n_2) = \frac{\sum_{t=T-\min(T,n_1)}^{\min(T,n_2)} R_k(\min(T - t, n_1), t)}{\min(T, n_1) + \min(T, n_2) + 1 - T},$$
$$n_1 + n_2 \geq T, \qquad k = 1, 2, \quad (3)$$

but all approaches attempted resulted in severe underestimates of response times. (Since the method of Newsom and Ward attempts this transformation, we are skeptical of its accuracy until it has been subjected to significant empirical evaluation.)

For these reasons, we accepted the expense of solving the network of Fig. 2. This allows us to simply use $\mu_k(n_1, n_2) = R_k(n_1, n_2)$. Herzog's method [26] does not easily extend to the network of Fig. 2. Since the underlying Markov process is not a two-dimensional birth and death process, Brandwajn's recent method [25] does not apply. However, Gauss-Seidel iteration and the related methods considered by Stewart [24] do apply. We choose to use Gauss-Seidel iteration. Implementation details are given in [27]. With non-preemptive priority there are no further difficulties. With FCFS there is the problem that the Markov states of the network of Fig. 2 must include information on the ordering of jobs in the memory queue, and thus the number of states is too large for numerical solution to be feasible. For this reason, we introduce another approximation, representing FCFS scheduling by random scheduling. With random scheduling, ordering information is not required for a Markov process representation, and numerical solution is feasible.

Besides scheduling, there is another issue of interest in the shared resource case which is a minor consideration in the dedicated resource case: The different job types may require different amounts of resource. So let us now consider T to be the number of abstract units of memory available and A_k, $k = 1, 2$, to be the number of units required by each type k job. When different types of jobs have different resource requirements, First-Fit (FF) scheduling is usually more interesting than FCFS. FF is the same as FCFS except that a satisfiable request is satisfied even when a job ahead in the queue must wait because its request is greater than the number of units currently available. This issue has little impact on the

difficulty of numerical solution of the network of Fig. 2. The more general memory organizations considered by Brown are easily incorporated in the iterative solution of the network of Fig. 2 [27].

Tables 4 and 5 summarize the cases considered in empirical evaluation of these methods. There are six groups of nine cases, with one case in each of those groups corresponding to one of the cases of Table 2. The first three groups have the same memory requirement for each job type. In the second three groups the second job type requires four times as much memory as the first. The first group has FCFS scheduling. The second and fifth groups have non-preemptive priority scheduling with type 1 jobs having higher priority. The third and sixth groups have non-preemptive priority scheduling with type 2 jobs having higher priority. The fourth group has FF scheduling. In Table 4 the column headed "App." gives the number of CPU seconds required for the approximate solution on a 370/168. Similarly, the column headed "Sim." gives the simulation CPU time. The four columns under "Ind. Errors" give the numbers of type-independent errors and severe errors for utilization (U), throughput (R), mean queue length (L), and mean queueing time (Q). (The maximum number of errors for a given measure is seven, one per queue.) Similarly the remaining four columns give the numbers of type-dependent errors. (The maximum number of errors for a given measure is fourteen, two per queue.) Table 5 has the same format as Table 3. It gives the approximation values and simulation confidence intervals for CPU utilization and mean response times. Measures with errors and severe errors are preceded by "e" and "s," respectively.

Generally speaking, the accuracy is good, though not as uniformly good as for the dedicated memory cases. The priority cases seem as free of errors as the dedicated memory cases, but case 14 (FCFS) and cases 42 and 45 (FF) have many errors. Even for these three cases the accuracy would likely be sufficient for many applications. The CPU time comparisons, though still quite favorable, are not as good because of the relatively "brute-force" iterative methods used for the solution of the network of Fig. 2.

5. Summary

We have shown how the "Norton's Theorem" approach to approximate solution of queueing networks with simultaneous resource possession can be extended to networks with heterogeneous jobs. We have empirically demonstrated the accuracy of the approach and shown that the approximation is typically two orders of magnitude less expensive than simulation. Both the accuracy and ex-

Table 4(a) Shared memory parameter combinations and error summary for First Come First Served memory scheduling and homogeneous memory requirements (1 unit).

Case	N_1	N_2	*Run time*			*Ind. errors*				*Dep. errors*			
			T	*App.*	*Sim.*	*U*	*R*	*L*	*Q*	*U*	*R*	*L*	*Q*
10	20	2	6	1	379	0	0	0	0	2	0	0	0
11	20	2	4	1	733	0	0	0	0	1	0	0	0
12	20	2	2	2	977	0	0	0	0	0	0	0	0
13	30	3	9	7	422	1	0	0	0	4	0	0	0
14	30	3	6	8	876	1	0	5	2	3	0	5	3
15	30	3	3	9	1394	0	0	0	0	0	0	0	0
16	40	4	18	20	669	0	0	0	0	2	0	0	0
17	40	4	12	23	1185	0	0	0	0	1	0	0	0
18	40	4	6	28	1343	0	0	0	0	0	0	0	0

Table 4(b) Shared memory parameter combinations and error summary for priority memory scheduling (type 1 has higher priority) and homogeneous memory requirements (1 unit).

Case	N_1	N_2	*Run time*			*Ind. errors*				*Dep. errors*			
			T	*App.*	*Sim.*	*U*	*R*	*L*	*Q*	*U*	*R*	*L*	*Q*
19	20	2	6	1	319	0	0	0	0	2	0	0	0
20	20	2	4	1	604	0	0	0	0	2	0	0	0
21	20	2	2	2	819	0	0	0	0	0	0	0	0
22	30	3	9	7	594	0	0	0	0	2	0	0	0
23	30	3	6	7	1076	0	0	0	0	1	0	0	0
24	30	3	3	8	834	0	0	0	0	1	0	1	1
25	40	4	18	18	806	0	0	0	0	2	0	0	0
26	40	4	12	21	778	0	0	0	0	1	0	0	0
27	40	4	6	26	680	0	0	0	0	0	0	0	0

Table 4(c) Shared memory parameter combinations and error summary for priority memory scheduling (type 2 has higher priority) and homogeneous memory requirements (1 unit).

Case	N_1	N_2	*Run time*			*Ind. errors*				*Dep. errors*			
			T	*App.*	*Sim.*	*U*	*R*	*L*	*Q*	*U*	*R*	*L*	*Q*
28	20	2	6	1	439	0	0	0	0	1	0	0	0
29	20	2	4	1	708	0	0	0	0	2	0	0	0
30	20	2	2	1	1394	0	0	0	0	0	0	0	0
31	30	3	9	6	634	0	0	0	0	2	0	0	0
32	30	3	6	7	1375	0	0	0	0	2	0	0	0
33	30	3	3	8	1507	0	0	0	0	0	0	0	0
34	40	4	18	17	613	0	0	0	0	2	0	0	0
35	40	4	12	19	1087	0	0	0	0	1	0	0	0
36	40	4	6	23	1623	0	0	0	0	0	0	0	0

Table 4(d) Shared memory parameter combinations and error summary for First Fit memory scheduling and heterogeneous memory requirements (type 1 jobs require 1 memory unit, type 2 jobs require 4 memory units).

Case	N_1	N_2	Run time			Ind. errors				Dep. errors			
			T	App.	Sim.	U	R	L	Q	U	R	L	Q
37	20	2	11	3	401	0	0	0	0	2	0	0	0
38	20	2	7	4	493	0	0	0	0	2	0	0	0
39	20	2	4	4	2048	0	0	0	0	2	0	0	0
40	30	3	14	23	1049	0	0	0	0	2	0	0	0
41	30	3	9	25	1518	0	0	0	0	2	0	0	0
42	30	3	5	21	3302	0	0	5	1	10	7	13	2
43	40	4	27	58	524	0	0	0	0	2	0	0	0
44	40	4	18	84	1731	0	0	1	1	0	0	1	1
45	40	4	9	87	2271	0	0	0	3	3	0	5	9

Table 4(e) Shared memory parameter combinations and error summary for priority memory scheduling (type 1 has higher priority) and heterogeneous memory requirements (type 1 jobs require 1 memory unit, type 2 jobs require 4 memory units).

Case	N_1	N_2	Run time			Ind. errors				Dep. errors			
			T	App.	Sim.	U	R	L	Q	U	R	L	Q
46	20	2	11	1	401	0	0	0	0	2	0	0	0
47	20	2	7	1	467	0	0	0	0	2	0	0	0
48	20	2	4	1	1533	0	0	0	0	2	0	0	0
49	30	3	14	7	854	0	0	0	0	2	0	0	0
50	30	3	9	8	1300	0	0	0	0	2	0	0	0
51	30	3	5	5	1081	0	0	0	0	2	0	0	0
52	40	4	27	19	624	0	0	0	0	2	0	0	0
53	40	4	18	23	726	0	0	0	0	2	0	0	0
54	40	4	9	17	426	0	0	0	0	2	0	0	0

Table 4(f) Shared memory parameter combinations and error summary for priority memory scheduling (type 2 has higher priority) and heterogeneous memory requirements (type 1 jobs require 1 memory unit, type 2 jobs require 4 memory units).

Case	N_1	N_2	Run time			Ind. errors				Dep. errors			
			T	App.	Sim.	U	R	L	Q	U	R	L	Q
55	20	2	11	3	437	0	0	0	0	2	0	0	0
56	20	2	7	4	2260	0	0	0	0	2	0	0	0
57	20	2	4	3	2519	0	0	0	0	2	0	0	0
58	30	3	14	21	1326	0	0	0	0	2	0	0	0
59	30	3	9	22	2956	0	0	0	0	2	0	0	0
60	30	3	5	5	989	0	0	0	0	2	0	0	0
61	40	4	27	53	409	0	0	0	0	2	0	0	0
62	40	4	18	74	2460	0	0	0	0	2	0	0	0
63	40	4	9	66	2797	0	0	0	0	2	0	0	0

pense should be acceptable in many applications, but some applications will require better accuracy or lower expense.

Though the focus has been on the memory contention model of Fig. 1, our discussion applies directly to other examples of simultaneous resource possession and to more than two simultaneously held resources, as in the homogeneous job case. Similarly, though we have assumed the network satisfies product form except for the simultaneous resource possession, the approach applies readily to non-product form networks which are amenable to flow-equivalence approximate solution.

Appendix: Bard's method

Bard's method for handling memory contention provides a very simple iterative approach [20]. The principal advantage of the approach is that it is very inexpensive. The method is generally less accurate than the methods we have described.

A fundamental assumption of the approach is that, if there is sufficient memory on the average for a given type of job, then that job never waits for memory. The types of jobs are partitioned into "trivial" and "non-trivial" types; it is assumed that trivial jobs never wait for memory. Our discussion here assumes all jobs are non-trivial jobs.

Let there be K job types. The following discussion assumes $A_k = 1$, $k = 1, \cdots, K$. It is simple to extend the discussion to avoid that assumption. From Little's Rule one can reasonably say that the mean number of type k jobs holding memory is

$$N_k \frac{H_k}{Q_{k,\text{Terminals}} + W_k + H_k}, \qquad k = 1, \cdots, K, \qquad \text{(A1)}$$

where H_k is the mean time type k jobs spend holding memory and W_k is the mean time type k jobs spend waiting for memory. Thus

$$\sum_{k=1}^{K} N_k \frac{H_k}{Q_{k,\text{Terminals}} + W_k + H_k} \leq T. \qquad \text{(A2)}$$

If (A2) results in a strict inequality, then it is assumed that there is no memory contention. This can clearly result in noticeable errors if the left-hand side of (A2) is nearly as large as the right-hand side. If (A2) results in an equality, then one must solve that equation for W_k, $k = 1, \cdots, K$. This solution depends on memory scheduling. Bard considers two cases, FCFS and "Fair-Share," a discipline used in the IBM VM/370 operating system. We ignore the Fair-Share discipline. The approach does not apply to FF

Table 5(a) Shared memory performance measures for First Come First Served memory scheduling and homogeneous memory requirements.

Case	U_{CPU}		Q_{Memory}		$Q_{1,Memory}$		$Q_{2,Memory}$	
10	0.62	(0.60, 0.63)	0.90	(0.87, 0.91)	0.75	(0.73, 0.76)	4.73	(4.34, 4.93)
11	0.61	(0.60, 0.62)	1.08	(1.05, 1.10)	0.93	(0.91, 0.95)	4.60	(4.46, 4.88)
12	0.54	(0.53, 0.54)	2.57	(2.46, 2.59)	2.45	(2.35, 2.47)	4.98	(4.85, 5.12)
13	0.84	(0.82, 0.84)	1.29	(1.20, 1.26)	1.08	(1.00, 1.04)	7.48	(6.95, 7.92)
14	0.82	(0.83, 0.85)	e1.62	(1.40, 1.47)	s1.42	(1.20, 1.26)	6.89	(6.87, 7.47)
15	0.71	(0.69, 0.71)	3.75	(3.60, 3.78)	3.59	(3.44, 3.62)	6.87	(6.79, 7.01)
16	0.96	(0.95, 0.96)	1.82	(1.78, 1.87)	1.50	(1.47, 1.54)	13.40	(12.41, 14.00)
17	0.95	(0.95, 0.96)	1.99	(1.94, 2.03)	1.68	(1.64, 1.71)	12.49	(11.63, 12.74)
18	0.90	(0.90, 0.90)	3.49	(3.44, 3.61)	3.24	(3.19, 3.35)	9.33	(9.23, 9.81)

Table 5(b) Shared memory performance measures for priority memory scheduling (type 1 has higher priority) and homogeneous memory requirements.

Case	U_{CPU}		Q_{Memory}		$Q_{1,Memory}$		$Q_{2,Memory}$	
19	0.62	(0.60, 0.64)	0.90	(0.88, 0.93)	0.75	(0.74, 0.77)	4.74	(4.39, 5.02)
20	0.61	(0.60, 0.62)	1.05	(1.03, 1.08)	0.91	(0.89, 0.93)	4.78	(4.39, 4.81)
21	0.50	(0.49, 0.50)	2.16	(2.11, 2.22)	1.91	(1.86, 1.95)	9.45	(9.35, 10.23)
22	0.84	(0.82, 0.84)	1.28	(1.24, 1.30)	1.07	(1.04, 1.09)	7.53	(6.78, 7.61)
23	0.82	(0.81, 0.82)	1.53	(1.50, 1.58)	1.31	(1.29, 1.35)	7.63	(7.27, 7.92)
24	0.59	(0.59, 0.60)	2.77	(2.78, 2.91)	2.32	(2.32, 2.43)	e22.49	(24.07, 27.30)
25	0.96	(0.95, 0.96)	1.82	(1.78, 1.86)	1.50	(1.47, 1.53)	13.41	(12.64, 14.15)
26	0.95	(0.95, 0.96)	1.95	(1.91, 2.00)	1.63	(1.59, 1.67)	12.85	(12.17, 13.53)
27	0.82	(0.81, 0.82)	2.67	(2.58, 2.70)	2.25	(2.19, 2.28)	19.37	(17.84, 2.13)

Table 5(c) Shared memory performance measures for priority memory scheduling (type 2 has higher priority) and homogeneous memory requirements.

Case	U_{CPU}		Q_{Memory}		$Q_{1,Memory}$		$Q_{2,Memory}$	
28	0.62	(0.62, 0.65)	0.90	(0.90, 0.94)	0.75	(0.75, 0.78)	4.72	(4.56, 5.11)
29	0.61	(0.60, 0.62)	1.09	(1.04, 1.09)	0.95	(0.91, 0.95)	4.51	(4.16, 4.53)
30	0.55	(0.54, 0.55)	2.79	(2.68, 2.81)	2.73	(2.62, 2.76)	3.78	(3.60, 3.75)
31	0.84	(0.84, 0.86)	1.29	(1.27, 1.34)	1.08	(1.06, 1.11)	7.46	(7.24, 8.11)
32	0.82	(0.81, 0.82)	1.66	(1.59, 1.67)	1.47	(1.41, 1.48)	6.65	(6.22, 6.67)
33	0.73	(0.73, 0.74)	4.25	(4.22, 4.44)	4.22	(4.19, 4.40)	4.77	(4.74, 4.98)
34	0.96	(0.95, 0.96)	1.82	(1.75, 1.83)	1.50	(1.45, 1.51)	13.40	(12.06, 13.78)
35	0.95	(0.95, 0.96)	2.01	(1.94, 2.04)	1.69	(1.64, 1.72)	12.39	(11.68, 12.97)
36	0.91	(0.91, 0.92)	3.83	(3.78, 3.97)	3.64	(3.58, 3.77)	7.84	(7.72, 8.18)

Table 5(d) Shared memory performance measures for First Come First Served memory scheduling and heterogeneous memory requirements.

Case	U_{CPU}		Q_{Memory}		$Q_{1,Memory}$		$Q_{2,Memory}$	
37	0.62	(0.60, 0.63)	0.91	(0.86, 0.90)	0.76	(0.72, 0.75)	4.72	(4.29, 4.82)
38	0.60	(0.61, 0.63)	1.02	(0.97, 1.01)	0.86	(0.80, 0.84)	5.18	(5.10, 5.71)
39	0.52	(0.52, 0.53)	1.96	(1.99, 2.09)	1.74	(1.78, 1.88)	7.52	(6.95, 7.28)
40	0.84	(0.83, 0.84)	1.34	(1.37, 1.44)	1.13	(1.17, 1.23)	7.36	(6.81, 7.32)
41	0.80	(0.80, 0.81)	1.69	(1.59, 1.67)	1.47	(1.37, 1.44)	7.64	(7.51, 8.07)
42	0.66	(0.66, 0.67)	s.2.48	(2.75, 2.84)	s2.12	(2.45, 2.53)	s15.34	(11.57, 12.02)
43	0.96	(0.96, 0.97)	1.83	(1.79, 1.88)	1.51	(1.48, 1.54)	13.34	(12.09, 14.13)
44	0.95	(0.95, 0.96)	e2.05	(2.21, 2.32)	s1.74	(1.92, 2.01)	12.28	(11.13, 11.87)
45	0.85	(0.78, 0.86)	2.71	(2.53, 2.61)	2.33	(2.18, 2.26)	e16.29	(14.13, 15.08)

Table 5(e) Shared memory performance measures for priority memory scheduling (type 1 has higher priority) and heterogeneous memory requirements.

Case	U_{CPU}		Q_{Memory}		$Q_{1,Memory}$		$Q_{2,Memory}$	
46	0.62	(0.60, 0.63)	0.91	(0.86, 0.90)	0.76	(0.72, 0.75)	4.72	(4.29, 4.82)
47	0.61	(0.60, 0.62)	0.97	(0.94, 0.99)	0.81	(0.79, 0.83)	5.28	(4.85, 5.47)
48	0.52	(0.52, 0.53)	1.85	(1.81, 1.90)	1.62	(1.58, 1.67)	8.22	(7.99, 8.47)
49	0.84	(0.82, 0.84)	1.32	(1.27, 1.33)	1.11	(1.06, 1.11)	7.43	(6.87, 7.54)
50	0.80	(0.80, 0.81)	1.55	(1.52, 1.60)	1.32	(1.30, 1.36)	8.21	(7.86, 8.61)
51	0.65	(0.65, 0.66)	2.19	(2.14, 2.25)	1.80	(1.76, 1.85)	18.44	(17.69, 19.44)
52	0.96	(0.95, 0.96)	1.83	(1.75, 1.84)	1.51	(1.46, 1.52)	13.35	(11.57, 13.32)
53	0.95	(0.95, 0.96)	1.97	(1.89, 1.98)	1.65	(1.59, 1.66)	12.77	(11.83, 13.18)
54	0.85	(0.83, 0.85)	2.23	(2.14, 2.25)	1.82	(1.74, 1.83)	2.96	(19.26, 21.77)

Table 5(f) Shared memory performance measures for priority memory scheduling (type 2 has higher priority) and heterogeneous memory requirements.

Case	U_{CPU}		Q_{Memory}		$Q_{1,Memory}$		$Q_{2,Memory}$	
55	0.62	(0.60, 0.62)	0.91	(0.87, 0.92)	0.76	(0.74, 0.77)	4.70	(4.27, 4.77)
56	0.60	(0.59, 0.60)	1.32	(1.25, 1.31)	1.18	(1.11, 1.17)	4.59	(4.52, 4.74)
57	0.54	(0.53, 0.54)	2.98	(2.75, 2.88)	2.92	(2.67, 2.81)	4.06	(4.10, 4.25)
58	0.83	(0.83, 0.84)	1.41	(1.34, 1.41)	1.21	(1.14, 1.20)	7.08	(6.78, 7.29)
59	0.81	(0.80, 0.81)	2.24	(2.14, 2.26)	2.08	(1.99, 2.10)	5.66	(5.55, 5.77)
60	0.65	(0.65, 0.66)	2.19	(2.13, 2.24)	1.80	(1.75, 1.84)	18.44	(17.58, 19.36)
61	0.96	(0.95, 0.97)	1.83	(1.81, 1.90)	1.51	(1.49, 1.56)	13.32	(12.51, 14.39)
62	0.95	(0.95, 0.95)	2.30	(2.20, 2.32)	2.01	(1.92, 2.02)	10.93	(10.61, 11.20)
63	0.88	(0.87, 0.88)	5.29	(5.08, 5.33)	5.23	(5.02, 5.26)	6.26	(6.17, 6.34)

Table A1 Shared memory performance measures (Bard's method).

Case	U_{CPU}		Q_{Memory}		$Q_{1,Memory}$		$Q_{2,Memory}$	
10	0.64	(0.60, 0.63)	e0.80	(0.87, 0.91)	s0.67	(0.73, 0.76)	e4.07	(4.34, 4.93)
11	0.64	(0.60, 0.62)	s0.80	(1.05, 1.10)	s0.67	(0.91, 0.95)	e4.07	(4.46, 4.88)
12	0.51	(0.53, 0.54)	e2.87	(2.46, 2.59)	e2.73	(2.35, 2.47)	s5.74	(4.85, 5.12)
13	e0.90	(0.82, 0.84)	s1.01	(1.20, 1.26)	s0.85	(1.00, 1.04)	s5.33	(6.95, 7.92)
14	e0.90	(0.83, 0.85)	s1.01	(1.40, 1.47)	s0.85	(1.20, 1.26)	s5.33	(6.87, 7.47)
15	0.70	(0.69, 0.71)	3.87	(3.60, 3.78)	3.71	(3.44, 3.62)	7.10	(6.69, 7.01)
16	1.00	(0.95, 0.96)	e1.63	(1.78, 1.87)	s1.34	(1.47, 1.54)	e11.51	(12.41, 14.00)
17	1.00	(0.95, 0.96)	s1.63	(1.94, 2.03)	s1.34	(1.64, 1.71)	11.51	(11.63, 12.74)
18	0.94	(0.90, 0.90)	e3.18	(3.44, 3.61)	e2.96	(3.19, 3.35)	e8.47	(9.23, 9.81)

or priority disciplines of the sort we have considered. With FCFS, Bard assumes that all job types have the same mean wait for memory, W.

The iteration proceeds by assuming values for H_k, $k = 1, \cdots, K$. When (A2) results in an equality, it becomes a nonlinear equation in a single unknown, which Bard suggests solving by Newton's method. Once W has been obtained, then one can obtain the mean number of each type of job holding memory from expression (A1). Those values can, in turn, be used to obtain new values for H_k, $k = 1, \cdots, K$. The iteration terminates, hopefully, when there is little change in the mean response times, $(W + H_k)$, $k = 1, \cdots, K$. Bard suggests using the mean time spent in service as the initial estimate for H_k, and that the initial value $W = 0$ be used with Newton's method. The iteration is not guaranteed to converge; Bard considers the iteration to have converged if successive values for the type-dependent mean response times do not vary by more than 5%.

We tried Bard's method as just described for cases 10–18 of Tables 4 and 5. Table A1 corresponds to Table 5 for these cases. The accuracy was significantly worse than with the Norton's Theorem approach, but still adequate for some applications. Bard's method is most attractive for larger problems than our test cases, *e.g.*, problems with more job types and larger populations, where the Norton's Theorem approach would be prohibitively expensive for most applications. Bard's method remains very inexpensive even for much larger problems.

References

1. C. H. Sauer and K. M. Chandy, *Computer System Performance Modeling*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.
2. *Computing Surveys* **10** (September 1978).
3. *Computer* **13** (April 1980).
4. J. R. Jackson, "Jobshop-like Queueing Systems," *Manage. Sci.* **10,** 131–142 (1963).
5. F. Baskett, K. M. Chandy, R. R. Muntz, and F. Palacios-Gomez, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *J. ACM* **22,** 248–260 (April 1975).
6. K. M. Chandy, J. H. Howard, and D. F. Towsley, "Product Form and Local Balance in Queueing Networks," *J. ACM* **24,** 250–263 (April 1977).
7. D. F. Towsley, "Queueing Network Models with State-Dependent Routing," *J. ACM* **27,** 323–337 (April 1980).
8. J. P. Buzen, "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *Commun. ACM* **16,** 527–531 (September 1973).
9. M. Reiser and S. S. Lavenberg, "Mean Value Analysis of Closed Multichain Queueing Networks," *J. ACM* **27,** 313–322 (April 1980).
10. K. M. Chandy and C. H. Sauer, "Computational Algorithms for Product Form Queueing Networks," *Commun. ACM* **23,** 573–583 (October 1980).
11. C. H. Sauer, "Computational Algorithms for State-Dependent Queueing Networks," *Research Report RC-8698*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, February 1981.
12. K. M. Chandy and C. H. Sauer, "Approximate Methods for Analysis of Queueing Network Models of Computer Systems," *Computing Surveys* **10,** 263–280 (September 1978).
13. C. H. Sauer and K. M. Chandy, "Approximate Solution of Queueing Models of Computer Systems," *Computer* **13,** 25–32 (April 1980).
14. A. Brandwajn, "A Model of a Time Sharing Virtual Memory System Solved Using Equivalence and Decomposition Methods," *Acta Informatica* **4,** 11–47 (1974).
15. R. M. Brown, "An Analytic Model of a Large Scale Interactive System Including the Effects of Finite Main Memory," Master's Thesis, Department of Computer Sciences, University of Texas, Austin, TX, 1974.
16. R. M. Brown, J. C. Browne, and K. M. Chandy, "Memory Management and Response Time," *Commun. ACM* **20,** 153–165 (March 1977).
17. P. J. Courtois, "Decomposability, Instabilities, and Saturation in Multiprogramming Systems," *Commun. ACM* **18,** 371–377 (July 1975).
18. P. J. Courtois, *Decomposability: Queueing and Computer System Applications*, Academic Press, Inc., New York, 1977.
19. T. W. Keller, "Computer Systems Models with Passive Resources," Ph.D. Thesis, University of Texas, Austin, TX, 1976.
20. Y. Bard, "An Analytic Model of the VM/370 System," *IBM J. Res. Develop.* **22,** 498–508 (September 1978).
21. P. A. Jacobson and E. D. Lazowska, "The Method of Surrogate Delays: Simultaneous Resource Possession in Analytic Models of Computer Systems," *Technical Report 80-04-03*, Department of Computer Science, University of Washington, December 1980.
22. B. R. Newsom and R. G. Ward, "Effects of Finite Memory on the Performance of a Large Scale Multiprogrammed Batch Computer System," *Proc. CMG X*, Dallas, 1979 (Computer Measurement Group, Inc., P.O. Box 26063, Phoenix, AZ 85068).
23. K. M. Chandy, U. Herzog, and L. Woo, "Parametric Analysis of Queuing Networks," *IBM J. Res. Develop.* **19,** 36–42 (January 1975).
24. W. J. Stewart, "A Comparison of Numerical Techniques in Markov Modeling," *Commun. ACM* **21,** 144–151 (February 1978).
25. A. Brandwajn, "An Iterative Solution of Two-Dimensional Birth and Death Processes," *Oper. Res.* **27,** 595–605 (May–June 1979).
26. U. Herzog, L. Woo, and K. M. Chandy, "Solution of Queuing Problems by a Recursive Technique," *IBM J. Res. Develop.* **19,** 295–300 (May 1975).
27. C. H. Sauer, "Numerical Solution of Some Multiple Chain Queueing Networks," *Research Report RC-8986*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, August 1981.
28. C. H. Sauer and E. A. MacNair, "Queueing Network Software for Systems Modeling," *Software—Practice and Experience* **9,** 369–380 (May 1979).
29. Charles H. Sauer, Edward A. MacNair, and Silvio Salza, "A Language for Extended Queuing Networks," *IBM J. Res. Develop.* **24,** 747–755 (November 1980).
30. D. L. Iglehart, "The Regenerative Method for Simulation Analysis," *Current Trends in Programming Methodology, Volume III: Software Modeling and Its Impact on Performance*, K. M. Chandy and R. T. Yeh, Eds., Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.
31. S. S. Lavenberg and C. H. Sauer, "Sequential Stopping Rules for the Regenerative Method of Simulation," *IBM J. Res. Develop.* **21,** 545–558 (November 1977).

Received February 9, 1981; revised June 2, 1981

The author is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.