Chapter 11

Processors

11.1. Introduction

Thus far we have considered only single CPU systems. We also have ignored the effects of the scheduling discipline that determines the order in which customers are served. In this chapter we will consider the representation of multiprocessors and scheduling disciplines.

In the realm of multiprocessor systems, an important distinction exists between *loosely-coupled multiprocessors* and *tightly-coupled multiprocessors.* In a loosely-coupled multiprocessor, the processors interact primarily through shared direct access storage devices. Since the processors operate essentially independently, they can be represented as separate service centers in a queueing network model, with different customer classes used to distinguish I/O operations originating from different CPUs. This approach was discussed in Chapter 10. In a tightly-coupled multiprocessor, the processors share main memory, and typically are under the control of a single operating system. Special techniques are required in building queueing network models of tightly-coupled multiprocessors; these techniques are the subject of Section 11.2.

Scheduling disciplines were ignored in the case of single class models (Chapter 6) because of two assumptions made there: that customers are indistinguishable (or "statistically identical") in their service demands, and that the expected *remaining* service time of a customer in service at a center does not depend on how much service the customer already has received. (The implication of this second assumption is that the expected time until the next customer completion at any particular center is not changed by removing one customer from service in order to serve another.) Given these two assumptions, system performance measures do not depend on the scheduling discipline used, as long as the processor is not idle when there is work to be done. The second assumption is violated, however, if the bursts of service required by a customer on successive visits to a processor vary widely in duration. Section 11.6 discusses an approach to modelling first-come-first-served (FCFS) scheduling when service bursts are highly variable.

In multiple class models, the situation is more complex. In Chapter 7 the following restrictions were placed on the scheduling disciplines used at queueing centers:

- The scheduling discipline cannot discriminate among customers based on class identity.
- If the scheduling discipline is FCFS, then the average time required to complete a customer in service must be independent not only of the amount of service it has acquired, but also of its class.
- If the scheduling discipline is *not* FCFS, then it must be one of a special group of disciplines that includes processor sharing (PS) and last come first served (LCFS). One important property of this group of disciplines is that each customer receives service immediately upon arrival at a center.

Under these restrictions, the performance estimates of a multiple class model are identical regardless of which of FCFS, PS, and LCFS scheduling is used at any center. Unfortunately, these scheduling disciplines do not adequately represent those used in many operating systems. In particular, class identity and the amount of acquired service often are used in making scheduling decisions. Separable models of such systems may not accurately reflect the relative performance of various workload components (classes). In Section 11.3 we suggest a way to model systems in which scheduling is done according to strict priorities among classes. In Section 11.4 we consider the more difficult case in which priorities are not based purely on class identity. Finally, in Section 11.5 we treat the case of FCFS scheduling when the service requirement per visit to the FCFS center differs from class to class.

11.2. Tightly-Coupled Multiprocessors

Tightly-coupled multiprocessor systems are in widespread use. These systems have two or more processors cooperating to complete work from a single shared queue.

It is easiest to view a tightly-coupled multiprocessor as a single service center, since in the system there is a single queue of jobs for all processors. The service rate of this center (i.e., the number of instructions delivered per time unit) is ideally the sum of the service rates of the individual processors. Consequently, the straightforward approach to modelling n tightly-coupled processors is to create a single center representing them in the model, and to divide the service demands of all customers at that center by n.

This technique provides a simple, first-cut modelling approach, but it ignores two important aspects of multiprocessors. The first aspect is that the total service rate of n processors can be significantly less than n times the rate of a single processor because of competition for software locks (such as those controlling access to the shared queue of jobs) and interference in accessing main memory. Thus, we need a more realistic assessment of the total processing power actually delivered by the multiprocessor. The second aspect is that the effective service rate of a multiprocessor is not constant, but depends on the number of jobs queued at the center. Consider a four processor system. Ideally, if four (or more) jobs desire service at the center, all four processors can be kept busy, and the effective service rate of the center is its maximum rate. However, if less than four jobs are queued at the center, some of the processors will be idle, and so the effective service rate will be reduced correspondingly.

The first of these problems, that of accounting for the interference of the processors with one another in estimating effective service rates, is best solved by using the results of benchmark studies of the configurations under consideration, such as those typically provided by trade journals and vendors. For example, such figures might indicate that an IBM 3033MP (a tightly-coupled dual processor) is roughly 1.7 times as powerful as a single 3033 processor when running a mixed TSO and batch workload under the MVS operating system. Since the power of a multiprocessor can vary significantly depending on the operating system run on it and the nature of the workload to be processed, standard estimates are not likely to be highly reliable. As in all cases where the input parameters are not known with high confidence, it is good practice to evaluate the model for several effective service rates representing a reasonable range, thereby assessing the sensitivity of the results to the parameter whose value is in question.

The second of these problems, that of accounting for variability in the effective service rate of the multiprocessor as a function of the number of jobs needing processor service, is solved easily using a flow equivalent service center. Figure 11.1 graphs effective service rate as a function of the queue length for a four processor system. Service rates increase with queue length until all four processors are busy, after which increasing the number of jobs contending for the processors does not result in any increase in effective service rate. The dashed line illustrates the ideal growth in service rate, and the solid curve represents the effect of contention. The flow equivalent service center used to represent the multiprocessor is parameterized by giving the effective service rates for each possible customer population that could be seen there. This set of population and service rate pairs is essentially a tabular representation of the curve shown in Figure 11.1.

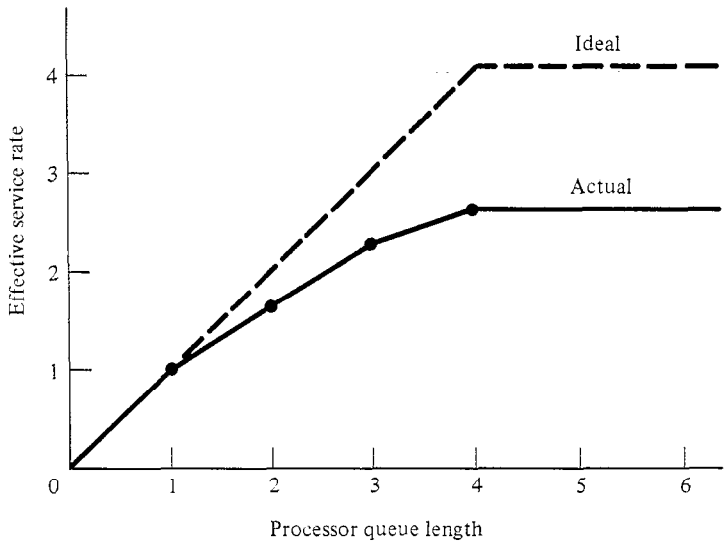

Figure 11.1 – Service Rate Function of a Four Processor System

11.3. Priority Scheduling Disciplines

In most current operating systems, processor scheduling disciplines are based on priorities. These priorities may be static (giving consistent preference to one workload component over another) or they may be dynamic (reflecting changing estimates of workload characteristics). Priority scheduling disciplines are not compatible with separable models. Since these disciplines can have a substantial effect on performance, it is important to be able to represent them. A number of approaches have been devised.

One approach was described as an example in Chapter 8. First, the I/O subsystem, which by itself was separable, was analyzed in isolation, and a multiple class flow equivalent service center was constructed. Then, a high-level model was defined that consisted of two centers: this FESC, and the priority scheduled CPU. Finally, the global balance technique was used to evaluate this model. This approach is quite accurate. Its drawbacks are, first, that it requires special purpose global balance software, and second, that because of the complexity of a global balance analysis it becomes infeasible for models with more than a few classes or customers, and for models with multiple priority scheduled centers.

Because of the difficulties in using the technique described in the last paragraph, another approach is required. The one we present here is

based on the mean value analysis technique. In practice, it has been found to be acceptably accurate, and is applicable even to very large models. Consider a model with C customer classes, each of which has a distinct priority at the CPU. (The generalization to several classes with equal priorities is straightforward.) For notational simplicity, assume that the classes are ordered so that higher numbered classes have priority over lower numbered classes. We develop an approximation to the residence time of class c customers at the CPU by considering successively the effects of jobs with lower, equal, and higher priorities than class c:

- *lower priority customers* (classes 1 through $c-1$)

  Because class c has preemptive priority over classes 1 through $c-1$, customers in these classes do not interfere with class c customers. Considering only these lower priority classes we obtain the following approximation to the CPU residence time of class c:

$$R_{c,CPU}(\vec{I}) \approx D_{c,CPU}$$

- *equal priority customers* (class c)

  Each class c customer arriving at the CPU must queue behind any other class c customers already there. Class c customers that arrive subsequently do not cause further delay. Accounting for both lower and equal priority classes we have:

$$R_{c,CPU}(\vec{I}) \approx D_{c,CPU}\left[1 + Q_{c,CPU}(\overrightarrow{I-1_c})\right]$$

  where $\overrightarrow{I-1_c}$ is the vector of workload intensities with one class c customer removed if class c is not transaction type (i.e., if class c is closed), and is the full workload intensity vector otherwise (i.e., if class c is open).

- *higher priority customers* (classes $c+1$ through C)

  An arriving class c customer must wait for all higher priority customers already in the queue. It also must wait for all higher priority customers that arrive while it is at the CPU. Because of this complication, it is not possible to estimate accurately the number of higher priority customers for which the class c customer must wait. Instead, we consider the servicing of higher priority customers to be "breakdowns" of the processor with respect to delivering service to the class c customers. Because of these breakdowns, more than $D_{c,CPU}$ time units are required for the class c customer in service to accumulate $D_{c,CPU}$ time units of service. In particular, since the CPU is busy $\sum_{j=c+1}^{C} U_{j,CPU}(\vec{I})$ of the time with higher priority customers, it takes $\dfrac{D_{c,CPU}}{1-\sum_{j=c+1}^{C} U_{j,CPU}(\vec{I})}$ time units for the currently selected class c

customer to complete. (For instance, once service is begun, it takes twice as long to complete on a processor 50% busy with higher priority customers than on a FCFS processor.) The final approximation for the residence time of class c, accounting for lower, equal, and higher priority classes, is thus:

$$R_{c,CPU}(\vec{I}) \approx \frac{D_{c,CPU}\left[1 + Q_{c,CPU}(\vec{I-1_c})\right]}{1 - \sum_{j=c+1}^{C} U_{j,CPU}(\vec{I})} \tag{11.1}$$

A solution technique could be constructed from the mean value analysis technique by substituting equation (11.1) for the standard residence time equation in Algorithms 7.1 or 7.2. However, rather than further complicating these basic algorithms each time we extend our modelling techniques, we prefer to build upon them, using the basic algorithms as subroutines in our extended algorithms. (We return to this concept of layered implementation in Chapter 16.)

In the case of priority scheduling, we can obtain the same results as we would obtain by replacing the residence time equation, by using the *shadow CPU* technique. This technique gets its name from the fact that the single priority scheduled CPU in the actual system is represented in the model by C FCFS service centers, each visited by one class. Let CPU_c denote the c-th shadow CPU, which is visited only by class c. The service demand at CPU_c is set equal to $\dfrac{D_{c,CPU}}{1 - \sum_{j=c+1}^{C} U_{j,CPU}(\vec{I})}$. It should be apparent that the residence time of class c at its shadow CPU is given by equation (11.1): the service demand inflation caused by higher priority classes is captured in the redefinition of the service demand at the shadow CPU, and the queueing for customers of class c but not other classes is a consequence of the FCFS scheduling used at the shadow CPU, plus the fact that only class c visits there. Thus, we have created a queueing network amenable to the analysis techniques of Chapter 7 that represents the effects of priority scheduling.

Algorithm 11.1 describes the shadow CPU technique more precisely. Because the CPU utilizations of the various classes are not known beforehand, it is necessary to employ iteration. Initially, the throughput of each class is estimated to be zero. This corresponds to estimating that the CPU utilization of each class is zero. The model is evaluated, yielding an improved estimate for the throughput, and thus the CPU utilization, of each class. New model inputs are calculated based on these improved estimates. The iteration continues until successive estimates of the throughput of each class are sufficiently close. Extension of Algorithm

1. Given a K center model with a priority scheduled CPU, create a $K+C-1$ center model by replacing the original CPU center with C FCFS shadow CPU centers, each of which will be visited by only one class. Assume that the classes are ordered so that higher numbered classes have priority over lower numbered classes. Initially, assume that the throughput of each class c, X_c, is equal to zero.
2. Iterate as follows:
   2.1. Estimate the CPU utilization of each class c as:
   $$U_{c,CPU} = X_c D_{c,CPU}$$
   where $D_{c,CPU}$ is the "real" CPU demand of class c.
   2.2. Set the service demand of each class c at the j-th shadow CPU to:
   $$D_{c,CPU_j} = \begin{cases} \dfrac{D_{c,CPU}}{1 - \sum_{k=c+1}^{C} U_{k,CPU}} & c=j \\ 0 & c \neq j \end{cases}$$
   2.3. Evaluate the shadow CPU model using either the exact or the approximate algorithms given in Chapter 7.

   Repeat Step 2 until successive estimates of the X_c for each class c are sufficiently close.
3. The final performance measures for the system as a whole and for every center except the CPU are obtained directly from the last iteration. At the CPU, the residence time of each class, $R_{c,CPU}$, and the queue length of each class, $Q_{c,CPU}$, are obtained directly. The utilization of each class, though, is obtained as $U_{c,CPU} = X_c D_{c,CPU}$. (The utilizations reported for the C shadow CPUs are meaningless because of the way in which the service demands have been inflated.)

Algorithm 11.1 – Priority Scheduling at the CPU

11.1 to the case in which several centers are priority scheduled is straightforward.

Table 11.1 shows the results of applying Algorithm 11.1 to a particular example. We consider a system with four disks and a priority scheduled

Model Inputs:

N_A = <varying> $Z_A = 10$ $N_B = 6$ $Z_B = 0$

	center				
	CPU	**Disk 1**	**Disk 2**	**Disk 3**	**Disk 4**
$D_{A,k}$	4	2	2	2	2
$D_{B,k}$	40	2	4	6	8

(all times are in seconds)

Class A Response Time:

solution technique	N_A				
	1	5	10	15	20
MVA	34.8	46.5	63.1	81.0	99.7
Algorithm 11.1	12.9	19.5	32.7	50.5	70.5
simulation	12.0	19.1	32.0	50.4	70.0

(all times are in seconds)

Table 11.1 – Priority Scheduling

CPU. There are two classes. Class A, which is of terminal type, has priority over class B, which is of batch type.

To assess the value of Algorithm 11.1 we would like to know whether its results are significantly better than those obtained by ignoring priority scheduling (i.e., by assuming that processor sharing is used). Unfortunately, we cannot determine exact performance measures for our example. Even though it has only five centers and two classes, it is too large to be analyzed using the global balance technique (described in Section 8.5.1). We have used simulation to obtain an estimate of the exact performance measures. As indicated in Section 8.5.2, simulation has two important drawbacks that make it less attractive than queueing network modelling for computer system analysis. First, the probabilistic nature of simulation causes the accuracy of its results to depend on the duration of the simulation. (For the duration used here, and in Sections 11.5 and 11.6, the error in the estimates obtained should be taken to be 5 to 10%.) Second, the computational expense of simulation is too great to allow it to be used regularly.

In the table we show the response time experienced by class A users for five different class A populations. The results obtained by ignoring the priority scheduling and applying mean value analysis directly are labelled "MVA" in the table, the results obtained by using Algorithm

11.1 are labelled "Algorithm 11.1", and the results obtained via simulation are labelled "simulation".

Comparing the results of MVA and Algorithm 11.1 illustrates the benefits of using Algorithm 11.1 rather than ignoring the priority scheduling. Comparing the results of Algorithm 11.1 and simulation illustrates the accuracy of Algorithm 11.1 for the specific example under consideration. Algorithm 11.1 will not always exhibit such close agreement to the results of simulation. Fortunately, though, the instances in which the algorithm may be unreliable are easy to identify. In most systems, priority scheduling is used to ensure that customers requiring short bursts of CPU service are not delayed excessively by customers requiring long bursts of CPU service. (Note that processor sharing is one step in this direction relative to FCFS scheduling, but that priority scheduling is one step further.) The technique presented in this section is designed to work well in this situation. It relies on the elongation of low priority service demands to reflect interruptions by high priority customers. This elongation is appropriate when service bursts of high priority customers are very short and very frequent relative to those of the low priority customers. However, whenever low priority service burst lengths are not significantly longer than high priority service burst lengths, the algorithm suggested in this section must be used with caution.

11.4. Variations on Priority Scheduling

While many operating systems permit specification of absolute priorities of the type discussed in the previous section, others support priorities of other natures. Two types of non-absolute priorities can be described as *biased processor sharing* and *goal-oriented scheduling.*

11.4.1. Biased Processor Sharing

Biased processor sharing describes a situation in which one class is favored over another by giving it longer bursts ("quanta") rather than by excluding the other class entirely when a customer of the higher priority class is present. Thus, a relative priority is associated with each class, and each customer receives service at a rate proportional to the relative priority of its class. For example, if the relative priorities of classes A and B are 2 and 1 respectively (a larger number indicating a higher priority), then with one customer of each class competing for service, the class A customer would progress at 2/3 the rate at which it would progress if alone at the center. With two class A customers and one of class B, each class A customer would progress at 2/5 of its full rate while the one class B customer would progress at 1/5 of its full rate.

An evaluation technique for this type of scheduling can be obtained by another modification of the residence time equation of the MVA algorithm:

$$R_{c,k}(\vec{I}) \approx D_{c,k}\left[\frac{\pi_c + \sum_{i=1}^{C} \pi_i Q_{i,k}(\overrightarrow{I-1_c})}{\pi_c}\right]$$

where π_i is the relative priority of class i. The quotient in parentheses is simply the inverse of the rate at which an individual class c customer receives service based on our expectation of the number of customers of each class at the center.

11.4.2. Goal-Oriented Scheduling

Goal-oriented scheduling differs from biased processor sharing in that dynamic scheduling priorities are used to ensure that each class attains specified performance objectives. For example, interactive users may be given general priority over a batch workload, subject to a constraint that batch throughput must have a certain minimum value. Such dynamic priorities are difficult to model in general, but creative use of transaction classes is helpful in some cases. For example, in the case described above, the model could initially give priority to the interactive class. If the solution indicates that the batch class attains its throughput goal, then no change to the model is needed. If the batch class fails to meet its throughput goal, however, we can assume that the goal-oriented scheduler would reduce the priority given to the interactive users enough to ensure the specified batch throughput. This can be reflected in the model by converting the batch workload to a transaction workload with its arrival rate set to the specified minimum throughput. For transaction classes, throughput is equal to arrival rate unless the system is saturated. Thus, the batch class is assured of the performance that it would attain under the goal-oriented scheduler, and the consequent degradation of service to the interactive class is represented.

11.5. FCFS Scheduling with Class-Dependent Average Service Times

If different classes have significantly different average service times per visit ($S_{c,k}$) at a FCFS center, our standard evaluation techniques from Chapter 7 may not provide acceptable accuracy. This situation is handled quite easily by another modification to the residence time equation of these techniques. The original form of the residence time equation is:

$$R_{c,k}(\vec{I}) = D_{c,k}\left[1+Q_k(\overrightarrow{I-1_c})\right] = V_{c,k}\left[S_{c,k} + S_{c,k}\,Q_k(\overrightarrow{I-1_c})\right]$$

Since all classes must have the same service time per visit at a FCFS center (in a separable network), we can think of this equation as a shortened form of:

$$R_{c,k}(\vec{I}) = V_{c,k}\left[S_{c,k} + \sum_{i=1}^{C} S_{i,k}\,Q_{i,k}(\overrightarrow{I-1_c})\right]$$

Simply substituting non-identical $S_{i,k}$ into the above equation provides an intuitively appealing evaluation technique for FCFS centers at which different classes have different average service times per visit: each class i customer found ahead of an arriving class c customer is multiplied by a class i service time. With this small change to one equation of the standard MVA algorithm, substantially more accurate solutions are obtained for models involving FCFS centers at which average service times differ from class to class.

An example is shown in Table 11.2. We consider a system with four disks and a CPU scheduled FCFS. There are two classes. Class A is of terminal type and class B of batch type. In the table we show the response time experienced by class A users for five different values of class A service time per visit at the CPU. We obtain results in three different ways: by ignoring the class-dependent average service times and applying mean value analysis directly ("MVA" in the table), by using the algorithm suggested in this section ("Section 11.5" in the table), and by simulating the system ("simulation" in the table).

The results show that the effect of class-dependent average service times can be pronounced, and that the algorithm suggested here yields good results for the example under consideration.

11.6. FCFS Scheduling with High Variability in Service Times

In the previous section we presented a solution technique for FCFS centers where the average service times per visit differ among the customer classes. This technique was necessary because of the restrictions required for a model to be separable (see Sections 7.2 and 7.5), and thus amenable to analysis using the standard algorithms of Chapter 7. In this section we present a technique that overcomes another restriction of separable networks, that imposed by the service time homogeneity assumption (see Section 7.5). This assumption states that the rate of completion of customers from any service center does not depend on the state of the model as a whole (i.e., the locations of the other customers).

Model Inputs:

$N_A = 10 \quad Z_A = 10 \qquad N_B = 6 \quad Z_B = 0$

	center				
	CPU	**Disk 1**	**Disk 2**	**Disk 3**	**Disk 4**
$S_{A,k}$	<varying>	1	1	1	1
$V_{A,k}$	8	2	2	2	2
$S_{B,k}$	2	1	1	1	1
$V_{B,k}$	20	2	4	6	8

(all times are in seconds)

Class A Response Time:

solution technique	$S_{A,CPU}$				
	2	1/2	1/8	1/32	1/128
MVA	250.1	63.1	26.8	23.7	23.4
Section 11.5	250.1	133.1	104.4	97.2	95.4
simulation	250.1	131.1	98.0	97.9	92.0

(all times are in seconds)

Table 11.2 – FCFS with Class-Dependent Average Service Times

In modelling most computer systems, any violation of this assumption does not result in significant error. Therefore, it is only in unusual situations that the technique to be presented need be employed. (We discourage superfluous use of the technique because it requires more parameter values than the simpler separable models, and so the parameterization effort is increased.)

As a rule of thumb, we can expect separable models to perform satisfactorily when the variability in service times per visit at each FCFS center is moderate, that is, when the average and standard deviation of service times are comparable. Centers for which the use of the technique will yield a noticeable improvement in accuracy are characterized by having most service bursts (service acquired in a single visit) be of comparable duration, with occasional bursts of much longer duration. As an example, in a batch system the CPU service quantum might be set very long to reduce context switch overhead; this could result in many short service bursts during file access, followed by a single long period of computation once the data has been acquired. In such a situation a separable model would not capture the effect on performance of the occasional very long service bursts, even if the average service time in the model was set to the measured average of the system. The effect of these long bursts is

to increase the amount of queueing that occurs in the system. Thus, a separable model will tend to give optimistic results when used in these situations.

As in other cases, we suggest a solution technique based on modifying the MVA residence time equation, then using the modified equation in the basic MVA iteration. Residence time consists of service time plus queueing time. Consider a class c customer arriving at service center k. Service time per visit ($S_{c,k}$) is an input parameter, and so presents no problem. Since we are considering FCFS centers, queueing time is required for all jobs already present at the center. The arriving job must wait on average $S_{i,k}$ time units for each class i customer found in the queue but not yet in service. Finally, the arriving customer must wait for the customer currently in service to finish. We can summarize this as:

$$R_{c,k}(\vec{I}) \approx V_{c,k}\left[S_{c,k} + \sum_{i=1}^{C} S_{i,k}\left[Q_{i,k}(\overrightarrow{I-1_c}) - U_{i,k}(\overrightarrow{I-1_c})\right] + \sum_{j=1}^{C} r_{j,k}\, U_{j,k}(\overrightarrow{I-1_c})\right]$$

where $r_{j,k}$ is the average time until completion of a class j customer found to be in service by a class c arrival at center k. The first term in this equation represents the inherent service requirement of the class c job. The second term approximates the total time spent waiting for customers in the queue (the $Q_{i,k}(\overrightarrow{I-1_c})$ term) but not in service (thus the $-U_{i,k}(\overrightarrow{I-1_c})$ term). Interpreting $U_{j,k}(\overrightarrow{I-1_c})$ as the proportion of time that an arriving class c customer finds a class j customer in service, the final term approximates the time spent waiting for the customer in service to complete.

This equation is the basis for an MVA-like analysis technique for models containing FCFS centers with high service time variability. The remaining problem is to estimate $r_{j,k}$, which often is called the *residual service time* of class j at center k. To do so, we assume that a class c job is equally likely to arrive at any point during the class j service interval (that is, class c arrivals occur at random with respect to class j service intervals). Even with this simplification, a reasonable choice for $r_{j,k}$ is not immediately apparent. Intuitively, one might guess $r_{j,k} = S_{j,k}/2$. In fact, however, this is an extreme value (representing the smallest possible residual service time) occurring only when the class j service times of all visits to center k are exactly equal. Under our assumptions, the residual service time is given by:

$$r_{j,k} = \frac{S_{j,k}}{2} + \frac{variance}{2S_{j,k}}$$

where *variance* is the variance in the service times per visit of class j at

center k. Thus, the actual residual can be any number at least as large as half the average service time (since it is possible for the variance to be any non-negative value). As an example, suppose class j experienced ten service bursts of length 1 for each burst of length 90. An arriving customer is then nine times as likely to arrive during the single long burst as during any of the short bursts. Thus, the residual service time is $(.1)(5) + (.9)(45) = 41$. In contrast, the average service time is $\frac{10}{11} 1 + \frac{1}{11} 90 = 9.09$. This surprising situation results from the fact that a customer is much more likely to arrive during a long burst than a short burst, even if many more bursts are short than long.

Table 11.3 presents an example of the use of this technique. We consider a system with four disks and a CPU. There is a single class of terminal type. In the table we show the response time experienced by users for five different degrees of variability in CPU service times. We obtain results in three different ways: by ignoring the high variability in CPU service times and applying mean value analysis directly ("MVA" in the table), by using the algorithm suggested in this section ("Section 11.6" in the table), and by simulating the system ("simulation" in the table).

The results show that the effect on performance of service time variability becomes more severe as this variability increases. The approach suggested in this section reflects the degradation in response time that occurs with increasing variability.

We note that this technique can be used whether the center we are considering has unusually high or low variance in service times per visit. While service time distributions with low variance also can be troublesome at FCFS service centers, their potential impact on model accuracy is more limited. Separable models tend to be slightly pessimistic for systems with low variance FCFS centers.

11.7. Summary

System configurations that include multiple processors or that use certain scheduling disciplines may require special techniques to obtain sufficiently accurate models. Tightly-coupled multiprocessors provide service at a total rate that depends on the number of jobs currently requiring CPU service. The set of processors is best represented as a single flow equivalent service center that provides service at a rate proportional to the number of busy processors, less a factor to account for interference among the processors. Loosely-coupled multiprocessors, on the other hand, require no such special treatment since each processor serves a separate job queue. Separate job classes can be used to distinguish jobs from different processors when they use shared I/O devices.

Model Inputs:

$$N = 10 \quad Z = 10$$

	center				
	CPU	**Disk 1**	**Disk 2**	**Disk 3**	**Disk 4**
S_k	0.5	1	1	1	1
V_k	8	2	2	2	2

(all times are in seconds)

Response Time:

solution technique	*variance* of S_{CPU}				
	.25	.5	1	2	4
MVA	32.6	32.6	32.6	32.6	32.6
Section 11.6	32.6	35.5	40.1	46.7	56.4
simulation	32.4	38.9	42.3	53.8	53.4

(all times are in seconds)

Table 11.3 – FCFS with High Variability in Service Times

Many operating systems use scheduling disciplines that are based on job class priorities, but priority scheduling is not compatible with separable models. Consequently, to obtain a model that can be validated, it may be necessary to employ a specialized technique for modelling priority scheduling. We have described a technique based on replacing the priority CPU by C "shadow" CPUs, each one visited by just one class. The service demand of each class at its shadow CPU is inflated to reflect the impact of higher priority classes. In some situations a different technique – based on hierarchical decomposition, a flow equivalent service center, and global balance – also may be applicable. Both of these techniques can be adapted to situations in which one, some, or all of the service centers are scheduled by priority. When priorities among classes are not absolute, it may be appropriate to model the discipline as biased processor sharing or goal-oriented scheduling. Techniques for treating these disciplines have been suggested.

Finally, FCFS scheduling also requires special treatment under some circumstances. If the average service requirement per visit to a center differs from class to class, then the model is not separable. Once again, a simple modification to the MVA algorithm produces good model solutions. Similarly, if there is high variability in the length of service times at each visit to a center, then FCFS scheduling cannot be accurately represented in a separable model. The high variability can be captured by

adapting the MVA solution technique, and by making further assumptions that allow estimates for the residual service time of jobs found in service by an arriving customer.

The techniques described in this chapter are useful for the specific circumstances in which they have been described. An equally important reason for presenting them, however, is that they are indicative of the approaches that must be creatively applied to achieve efficient and accurate solutions to non-separable models.

11.8. References

Sauer and Chandy were the first to use flow equivalent service centers and a global balance solution of a two center model to evaluate non-separable models, including ones involving priority scheduling [Sauer & Chandy 1975]. They discuss other techniques for evaluating non-separable models elsewhere [Chandy & Sauer 1978; Sauer & Chandy 1980].

Bard first demonstrated the flexibility of the basic MVA algorithm in adapting to non-separable models, treating both priority models and models in which different classes have distinct average service requirements per visit to an FCFS center [Bard 1979]. Bard also has described a modelling approach capable of treating the dynamic priority scheduling used in IBM's VM/370 operating system [Bard 1981].

The shadow CPU technique described in Section 11.3 was developed by Sevcik [1977]. His approach involved identifying separable models that provide optimistic and pessimistic bounds on the performance of a (non-separable) model with a priority center.

The MVA-based approach to modelling high service time variability was proposed by Reiser and Lavenberg [1978]. An alternative approach is based on global balance and Cox's *method of stages* representation [Cox 1955]. Cox demonstrated that arbitrary service time distributions can be approximated as closely as desired by using a sufficient number of exponentially distributed stages with probabilistic selection. Sevcik, Levy, Tripathi, and Zahorjan describe three-parameter method of stages representations for both high variability and low variability distributions [Sevcik et al. 1977]. With these three-parameter representations, it is possible to match two characteristics (typically the mean and variance) of an arbitrary distribution. Lazowska has shown that more accurate models are obtained by matching the mean and some percentile (say the 90th) than by matching the mean and variance [Lazowska 1977]. Lazowska and Addison provide a technique for determining a method of stages

representation that matches the mean and an arbitrary number of percentiles of an arbitrary distribution [Lazowska & Addison 1979].

The simulation results reported in Tables 11.1, 11.2, and 11.3 were obtained from IBM's Research Queueing Package [Sauer et al. 1982].

[Bard 1979]
Yonathan Bard. Some Extensions to Multiclass Queueing Network Analysis. In M. Arato, A. Butrimenko, and E. Gelenbe (eds.), *Performance of Computer Systems.* North-Holland, 1979.

[Bard 1981]
Yonathan Bard. A Simple Approach to System Modelling. *Performance Evaluation 1,3* (November 1981), 225-248.

[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), 281-317.

[Cox 1955]
D.R. Cox. A Use of Complex Probabilities in the Theory of Stochastic Processes. *Proc. Cambridge Philosophical Society 51* (1955), 313-319.

[Lazowska 1977]
Edward D. Lazowska. The Use of Percentiles in Modeling CPU Service Time Distributions. In K.M. Chandy and M. Reiser (eds.), *Computer Performance.* North-Holland, 1977, 53-66.

[Lazowska & Addison 1979]
Edward D. Lazowska and Clifford A. Addison. Selecting Parameter Values for Servers of the Phase Type. In M. Arato, A. Butrimenko, and E. Gelenbe (eds.), *Performance of Computer Systems.* North-Holland, 1979, 407-420.

[Reiser & Lavenberg 1978]
Martin Reiser and Stephen S. Lavenberg. Mean Value Analysis of Closed Multichain Queueing Networks. Report RC-7023, IBM T.J. Watson Research Center, March 1978.

[Sauer & Chandy 1975]
Charles H. Sauer and K. Mani Chandy. Approximate Analysis of Central Server Models. *IBM Journal of Research and Development 19,3* (May 1975), 301-313.

[Sauer & Chandy 1980]
C.H. Sauer and K. Mani Chandy. Approximate Solution of Queueing Models. *IEEE Computer 13,4* (April 1980), 25-32.

[Sauer et al. 1982]
Charles H. Sauer, Edward A. MacNair, and James F. Kurose. The Research Queueing Package, Version 2: Introduction and Examples. Report RA 138, IBM T.J. Watson Research Center, 1982.

[Sevcik 1977]
Kenneth C. Sevcik. Priority Scheduling Disciplines in Queueing Network Models of Computer Systems. *Proc. IFIP Congress '77* (1977), 565-570.

[Sevcik et al. 1977]
Kenneth C. Sevcik, Allan I. Levy, Satish K. Tripathi, and John Zahorjan. Improving Approximations of Aggregated Queueing Network Subsystems. In K.M. Chandy and M. Reiser (eds.), *Computer Performance.* North-Holland, 1977, 1-22.

11.9. Exercises

1. Consider a single class model of a dual processor system. The service demand at the CPU is 8 seconds (with each processor providing a portion of this service) and the service demands at each of the four disks are 2 seconds. The single customer class is of terminal type, with $Z = 20$ seconds.
   a. Compare the results obtained by modelling the dual processor as a single fast processor (with a service demand of 4 seconds) to the results obtained by using the FESC approach of Section 11.2 (with service rates of 0.125 with one customer in the queue, and 0.250 with more than one customer in the queue). Obtain solutions for populations of 5, 10, and 20 online users. (Use the MVA implementation of Chapter 18, extended to accommodate FESCs and terminal classes.)
   b. What do your solutions for the three population sizes indicate about the accuracy of the "single fast processor" approach in (a)? How well would you expect this approach to work if the configuration contained four processors rather than two?
2. Section 11.3 developed a technique for modelling preemptive priority CPU scheduling. Using this as a basis, develop a technique for modelling non-preemptive priority scheduling. Under non-preemptive priority, a job in service at the CPU receives a full service burst, even if a higher priority job arrives during that burst. When the service burst completes, the highest priority waiting job is selected for the next service burst.

3. Consider a simple interactive computer system consisting of a CPU and four disks. Assume that the disks are scheduled FCFS, and that users can choose their I/O block size: the number of bytes transferred between a file and main storage on each access. Measurements of the system show that 75% of the users choose block sizes resulting in service times per disk visit of 32 milliseconds, and 25% choose sizes resulting in service times per disk visit of 44 milliseconds.
   a. Suppose that there are a total of 24 online users divided into two classes based on blocksize. Both classes have 20 second think times, and have interactions that require 4 seconds of CPU service and an average of 100 accesses to each of the four disks. Use the technique of Section 11.5 to estimate response times for each class.
   b. Using the throughput values obtained from (a), compute the average service time per I/O operation at each disk. Use this value to construct a model of the system with a single class of "average" users. This model can be evaluated using standard mean value analysis techniques.
   c. Compute the overall average response time in the two class model of (a). (Remember that the response times of the classes must be weighted by their throughputs.) Compare your result to the response time obtained in (b). What does this tell you about the effect on system performance of FCFS scheduling with class-dependent service times?
   d. Repeat (a) through (c) under that assumption that 75% of the users have disk service times of 12 milliseconds, and 25% have disk service times of 116 milliseconds. Compare your results to those obtained earlier. What does this tell you about the importance of reflecting service time variability in models of computer systems?
   e. Returning to the single class model, use the technique of Section 11.6 to model the high service time variability of an "average" job at each disk. To do so, you will need to estimate the variance of the service times at the disks. If proportion p of the total accesses require S_1 time units and proportion $1-p$ require S_2 time units, then the average service time S is equal to $pS_1 + (1-p)S_2$, and a reasonable estimate of the variance in service times is:

$$\textit{variance} = p(S_1 - S)^2 + (1-p)(S_2 - S)^2$$

   Calculate response times for the original set of disk service times and the modified set of (d), and compare these to the results obtained earlier. How do you account for the differences in the various estimates?

4. Discuss the treatment of scheduling disciplines in single class, separable queueing network models.
5. Discuss the treatment of scheduling disciplines in multiple class, separable queueing network models.
6. We have considered FCFS scheduling in four contexts: single class separable models, multiple class separable models, single class with high variability in service times, and multiple class with class-dependent average service times. Compare and contrast these.