Chapter 8

Flow Equivalence and Hierarchical Modelling

8.1. Introduction

The models studied in previous chapters were simple both in their construction and in the techniques required for their evaluation. Often it is useful to construct more sophisticated models so that additional details of the computer system may be represented. In this chapter we discuss a technique for doing so, *hierarchical modelling*. Hierarchical modelling is the process of partitioning a large model into a number of smaller submodels. Each of these submodels then is evaluated, and the individual solutions are combined to obtain the solution of the original model. The recombination is performed using a special type of service center called a *flow equivalent service center (FESC)*.

Consider the model shown in Figure 8.1, which represents two single-CPU systems with a shared I/O subsystem. In the general case, there is an arbitrarily defined subsystem, called the *aggregate*, which interacts with the other service centers in the network, called collectively the *complement* or *complementary network*. The aggregate itself may or may not be representable as a network of service centers. In the case of this example, the complement represents the CPUs, while the aggregate represents the complex I/O subsystem. A key step in the hierarchical approach is to replace the entire aggregate by a single service center that mimics its behavior, thus reducing the size of the network to be solved.

From the perspective of the service centers in the complement, the aggregate can be thought of as a black box whose behavior is characterized by the residence time there (i.e., the time interval from when a customer enters the aggregate until that customer departs the aggregate) and by the rate and pattern by which customers leave the aggregate to return to the complement (i.e., the departure process of the aggregate). As long as customers experience an appropriate delay at the aggregate, and the departure process of the aggregate is correct, the service centers in the complement are unaffected by the actual construction of the aggregate. Therefore, any representation of the aggregate that results in appropriate inter-departure times is sufficient to obtain the solution of the network

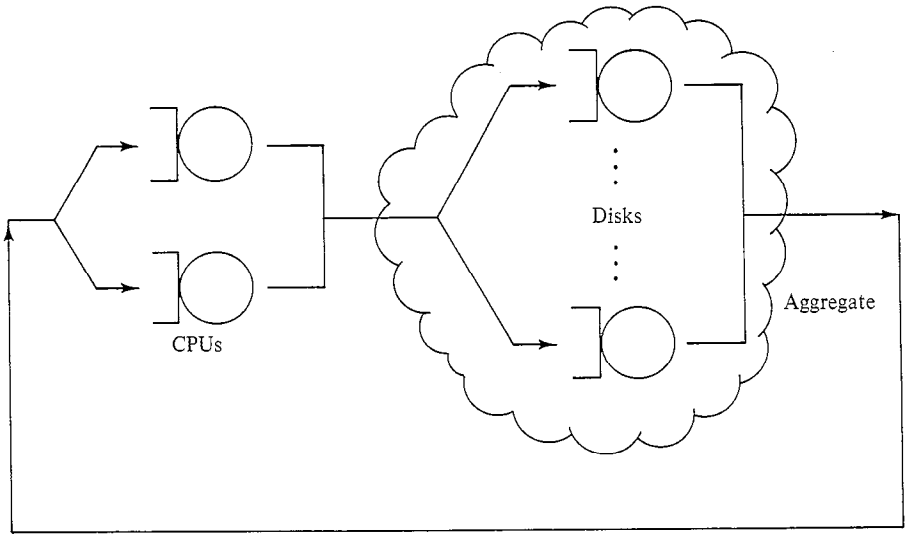


Figure 8.1 – Example Loosely-Coupled Multiprocessor Model

(with respect to the service centers in the complement). In particular, the performance measures obtained for the complementary network will be the same regardless of whether the aggregate is represented as a large number of service centers or as a single service center.

It is this realization that leads to the concept of flow equivalent service centers. An FESC is a single service center that, from the point of view of the complementary network, behaves identically to the aggregate itself. This means that the FESC must (minimally) cause the same average delay to customers passing through it as those customers would experience had they actually proceeded through the detailed representation of the aggregate. (In general, for an FESC to be exact, it must mimic the actual distribution of interdeparture times from the aggregate, not just the average. However, such detailed FESCs are too cumbersome to be of practical use, so we limit ourselves to FESCs that match only average residence time and throughput.) Since the FESC is a single service center, while the detailed representation of the aggregate presumably is much more complex, the use of FESCs is attractive because it leads to much simpler models.

FESCs are the keys to hierarchical modelling. Hierarchical modelling (often called *hierarchical decomposition*) is the process of modelling a system using multiple levels of models. The model at the highest level, level 0, consists of a number of FESCs, each of which represents some portion of the computer system being modelled. The level below that,

level 1, consists of a number of models, each a more detailed representation of a subsystem represented in level 0 as an FESC. Each of the level 1 models itself may contain FESCs. In general, the characteristics of the FESCs at level l are determined by solving models at level $l+1$, until finally some level is reached at which all models are fully detailed, i.e., contain no FESCs. Figure 8.2 shows a possible decomposition scheme. (Notationally, FESCs are distinguished by an arrow through the server, suggesting variability.)

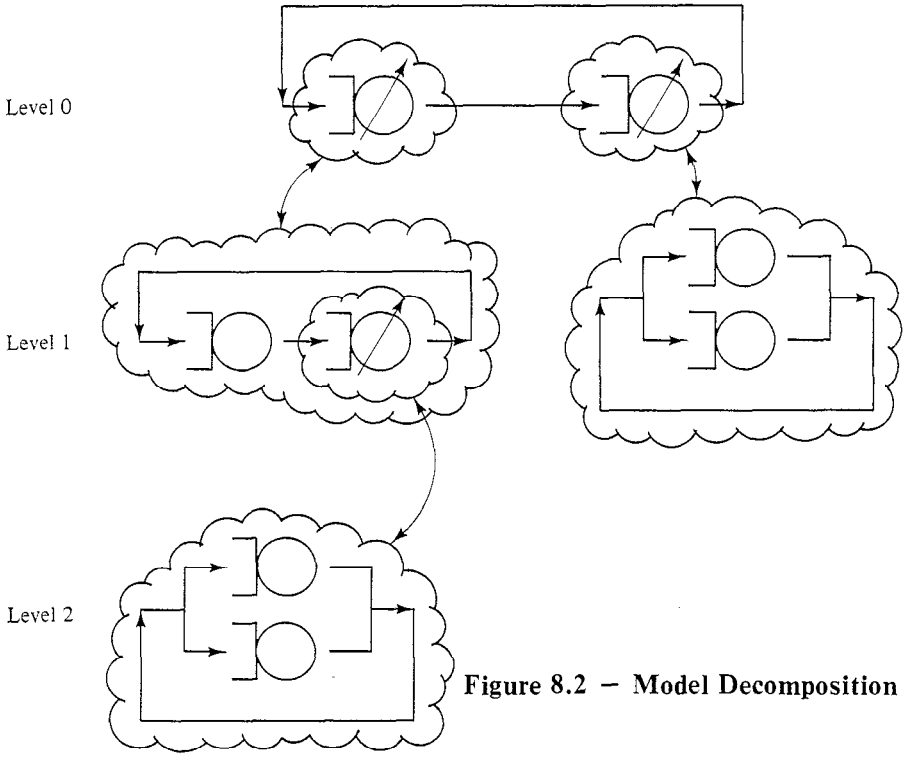


Figure 8.2 – Model Decomposition

Although the definition of the models normally proceeds from level 0 to level L, the evaluation of the models must occur in the opposite direction, i.e., from level L to level 0. Eventually the level 0 model is evaluated, and performance projections for the computer system being modelled are obtained from its solution.

There are two key requirements in hierarchical modelling beyond the original need to define the levels of models. The first is to find a suitable structure for FESCs. Our goal is to create a single service center that can replace an entire subsystem. Thus, we expect this center to be more complicated than the service centers we have seen so far, which represent only single resources. Intimately related to the problem of finding a suitable representation for the level l FESCs is the problem of obtaining

parameter values for them from the submodels at level $l+1$. These issues are considered in Sections 8.2 and 8.3.

The second requirement of the hierarchical modelling process is to evaluate models containing FESCs. As mentioned above, we should expect FESCs to be more complicated than the types of centers we have seen so far. Correspondingly, we should expect the solution techniques required to evaluate models containing them to be more complicated. This issue is addressed in Section 8.4.

8.2. Creating Flow Equivalent Service Centers

In general, it is not possible to find FESCs that produce exact results for the complementary network. However, reasonably accurate approximations can be obtained. Figure 8.3 shows a typical situation in which an FESC might be used. The enclosed subsystem (the aggregate) would be replaced by the FESC.

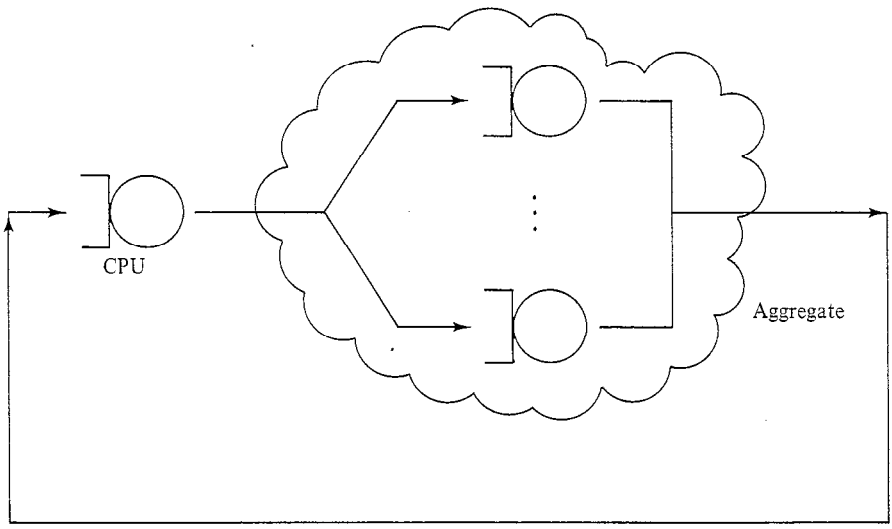


Figure 8.3 – Example Application of an FESC

The purpose of the FESC is to mimic the behavior of the aggregate. This behavior, as viewed by the complementary subnetwork, is the flow of customers out of the aggregate and into the complement. An approximation for this flow can be obtained by making the *decomposability assumption* that the average rate at which customers depart the aggregate depends only on the state of the aggregate, where the state is defined by the customer population within the aggregate. Thus, the state is

independent of the placement of the customers at the various service centers. (For example, the state of an aggregate might be (2 class A , 1 class B). The total number of customers of each class is represented, but information about the location of each customer in the aggregate is ignored.) An aggregate therefore can be defined completely by a listing of its throughputs as a function of its possible customer populations.

The assumption that the output rate of the aggregate depends only on the customers in it implies the assumption that the aggregate achieves *local equilibrium* between successive interactions with the complement. Local equilibrium means that the behavior of the aggregate is independent of its starting condition. This situation occurs if, after an arrival to the aggregate, many transitions of customers between service centers in the aggregate occur before another arrival from the complement takes place. Local equilibrium is most likely achieved when the service centers in the aggregate all have service rates that are considerably faster than the service rates of the centers in the complement.

It is desirable that the aggregate achieve local equilibrium because in that case the average departure rate from the aggregate with a given population in it will be nearly the equilibrium throughput, regardless of the initial placement of those customers. This is exactly the assumption made in reducing the aggregate to a single service center whose state is described entirely by the number of customers present. If the aggregate did not achieve equilibrium, its output rate would depend on its initial configuration of customers, and so the single server representation would be deficient.

Flow equivalent service centers are represented in queueing network models using *load dependent service centers.* A load dependent service center can be thought of as a service center whose service rate (the reciprocal of its service time) is a function of the customer population in its queue. For instance, a delay center can be thought of as a load dependent service center that has service rate μ with one customer in the queue, and service rate $n\mu$ with n customers in the queue (in a single class model). In contrast, a queueing service center is *load independent*: it has service rate μ regardless of the number of customers in its queue.

An FESC for an aggregate is a load dependent service center with service rates $\mu_c(\vec{n})$ equal to the throughputs $X_c(\vec{n})$ of the aggregate for all populations $\vec{n}$ and classes c. (We will discuss methods for obtaining these rates in Section 8.3.) Because the FESC mimics the behavior of the aggregate, it can be used to replace the detailed description of the aggregate in the model with little effect on the performance measures obtained.

For single class models, a state of an aggregate is described simply by the number of customers anywhere within it, since customers are indistinguishable. A flow equivalent service center is formed by calculating

throughputs $X(n)$ of the aggregate as a function of the number n of customers in the aggregate. These are used to create a load dependent service center with service rates $\mu(n) = X(n)$.

In the case where the workload is transaction type, a rather subtle problem can occur with the specification of the FESC. For these models, there is no limit to the number n of customers that might exist in the aggregate. Thus, an infinite number of throughput values seem to be required to specify the FESC. While this is the case in theory, in practice the situation is less bleak. Because real computer systems do not experience unbounded numbers of jobs in their queues, only a finite (and usually small) number of rates are required even for transaction type classes. Typically, distinct rates are specified for all n less than some given number n^* (which depends on the computer system being modelled). Rates for all larger n are then assumed to be equal to the rate with n^* customers. FESCs that have rates of this sort are said to have *limited load dependent* behavior. We will see specific applications of limited load dependence in Part III of this book.

In applying FESCs to multiple class models, the state of an aggregate is defined by a vector $\vec{n} \equiv (n_1, \ldots, n_C)$ giving the number of customers of each class present. Thus, the flow equivalent service center corresponding to a specific aggregate is the load dependent service center with output rate for class c, $\mu_c(\vec{n})$, equal to $X_c(\vec{n})$. Since the output rate of the FESC for each class must equal that of the aggregate, the "scheduling discipline" at multiple class FESCs cannot be a traditional one. (For example, if an FESC were scheduled FCFS, only the class currently in service at the FESC would exhibit the proper output rate, since all other classes would have output rates of zero.) Instead, an artificial scheduling discipline, called *composite queueing*, is used so that all classes receive service at once. One can think of the FESC as having C distinct queues, one for each customer class. These queues are served in parallel, with the class c queue being served at rate $\mu_c(\vec{n})$ when the population of the C queues is given by $\vec{n} \equiv (n_1, \ldots, n_C)$.

As with single class models, specifying rates for an FESC in a network that contains transaction type job classes can present problems in theory, because of the apparently unbounded number of rates required. In practice, though, FESCs with limited load dependent behavior are sufficient, and so models with transaction type classes pose no real problems.

A problem associated with multiple class FESCs that does not arise in the single class case is that the number of populations for which throughputs must be determined grows very quickly with the number of classes. In particular, $C\prod_{c=1}^{C}(N_c+1)$ throughputs are required for a network with a (closed) population of N_c class c customers (a throughput

for each of the C classes, for each of the $\prod_{c=1}^{C}(N_c+1)$ possible aggregate populations). A network with five classes of ten customers each, for instance, requires nearly one million distinct throughputs. Fortunately, this problem can be dealt with in some cases by choosing an appropriate method for calculating the necessary load dependent throughputs (see Section 8.3).

It is important to keep in mind that while the hierarchical modelling process appears to give an exact representation of the model, in general it is only an approximation. The approximation arises in describing an entire subsystem by a single service center. In doing so, information regarding the placement of customers at the centers of the subsystem is lost, and so the FESC does not have sufficient information to mimic the subsystem exactly. In many situations, however, the resulting inaccuracy is negligible.

8.3. Obtaining the Parameters

The parameters required to specify an FESC are the load dependent service rates for each class as functions of the possible queue populations. As indicated previously, the rates for level l models generally are obtained from the solution of the corresponding level $l+1$ models. However, there are a number of different ways in which a level $l+1$ model can be evaluated:

- *measurements* — In some cases, it may be possible to observe the subsystem that is to be aggregated, and to obtain measurements of its throughput as a function of the number of customers present. For instance, one might measure the throughput of a channel/string pair as a function of the number of outstanding requests to that string. These measured throughputs then could be used directly to set the service rates of an FESC.
- *queueing network models* — The level l FESC might be representable at level $l+1$ as a queueing network consisting of load independent service centers (and possibly some FESCs with service rates set by solutions of lower-level models). This level $l+1$ model can be evaluated analytically, and the throughputs predicted from its solution used to set the service rates of the level l FESC.
- *simulation* — If some aspects of the aggregate make it difficult to evaluate analytically, a simulation of the aggregate can be performed to obtain the required load dependent throughputs.

- *special purpose analytic methods* — Models peculiar to a particular subsystem, such as a complex I/O subsystem, might be developed and solved analytically. The outputs of these models could be load dependent throughputs, which then would be used to define the FESC required in the next higher-level model.

In most cases we advocate the use of queueing network models for establishing the parameters of FESCs, for the same reasons that we advocate their use in general: a combination of reasonable accuracy and ease of use. Additionally, this approach has the overwhelming advantage of producing all $C\prod_{c=1}^{C}(N_c+1)$ rates required to parameterize the FESC with a single solution of the low-level model. (Remember that the exact MVA solution algorithm produces solutions for all populations from 0 to $\vec{N}$ as a by-product of obtaining the solution at population $\vec{N}$.)

Having obtained the parameters of the level l FESCs, we now must evaluate the level l model. As this model is simply one of the low-level models defining a level $l-1$ FESC, it is clear that we can use any of the preceding techniques to perform this analysis. However, for the reasons outlined above, it generally is the case that the second method (queueing network models) is used. In the next section we look in more detail at the process of applying this technique.

8.4. Solving the High-Level Models

The most obvious approach to evaluating high-level models is to apply the analytic techniques developed in previous chapters. In Chapter 20 we present extensions to the MVA solution technique that allow the efficient evaluation of networks containing load dependent service centers. Unfortunately, this approach is applicable only to separable queueing network models. *Non-separable* high-level models can arise when some non-separable aspect of the original model (such as a priority scheduled service center) is represented directly in the high-level model, or when the load dependent service centers have arbitrary service rate functions.

For the moment, let us assume that the original network to be analyzed is separable, so that the first of these two problems cannot arise. In this case, if we wish to evaluate the higher-level model using efficient analytic techniques, we require certain restrictions on the load dependent service rates of each FESC. In particular, it must be possible to describe the service rates of each FESC by a C dimensional matrix $g[0{:}N_1, 0{:}N_2, \ldots, 0{:}N_C]$, such that the service rate of class c with population $\vec{n}$, $\mu_c(\vec{n})$, is equal to $\dfrac{g[n_1, \ldots, n_c-1, \ldots, n_C]}{g[n_1, \ldots, n_C]}$, with the initial

condition that $g[0, \ldots, 0]=1$. A simple example of plausible throughput rates for a two-class aggregate that violate this condition is:

$$\begin{aligned} \mu_A(n_A=1, n_B=0) &= 1/2 \\ \mu_B(n_A=0, n_B=1) &= 1/3 \\ \mu_A(n_A=1, n_B=1) &= 3/10 \\ \mu_B(n_A=1, n_B=1) &= 2/9 \end{aligned}$$

The first two rates require that $g[1,0]=2$ and $g[0,1]=3$ (remembering that $g[0,0]$ is equal to 1). The last two rates are incompatible, since the rate for class A requires that $g[1,1]$ be 10, while the rate for class B requires that it be 9.

While general techniques for estimating the service rates of FESCs do not lead to separable higher-level models, analyzing the lower-level models as separable networks (the second approach of Section 8.3) is guaranteed to do so. Based on this fact, an efficient strategy for use in the hierarchical modelling of separable networks is summarized as Algorithm 8.1. While the primary motivation for this strategy is its low computational requirement, it happens that when the original model is separable, this algorithm produces the exact solution.

In cases where the original model is not separable, Algorithm 8.1 must be modified slightly. If the non-separable aspect of the model is included in one of the lower-level models, then the step of the algorithm that solves that submodel must be modified, as the MVA solution technique is not applicable. Similarly, since the throughputs obtained from a non-separable submodel do not result in a separable FESC, the step of the algorithm dealing with the solution of the high-level model must be modified. If the non-separable aspects of the original model do not appear in any low-level models, but only in the higher-level model, only the step dealing with the solution of this model must be altered. An approach to solving non-separable models that can be used in place of MVA in applying Algorithm 8.1 is given in Section 8.5. That approach results in approximate solutions of the original model. However, experience has shown that such approximations usually are quite accurate.

8.5. An Application of Hierarchical Modelling

To this point we have been concerned with separable queueing network models. The principal advantage of separable networks over more general networks is that their solutions can be obtained very quickly. However, the conditions required for separability impose some restrictions that at times can result in insufficiently accurate models. There are three approaches that can be taken in such a case. One is to combine the solutions of a number of separable networks (possibly with some iteration

Given a closed, separable model with K centers and population $\vec{N}$, let centers 1 through A represent the aggregate, and centers $A+1$ through K the complement.

1. Create a low-level model by setting the service demands of centers $A+1$ through K to zero for all classes. This is equivalent to creating a model with centers 1 through A.
2. Evaluate this (separable) model with population $\vec{N}$, using the exact MVA solution technique. Obtain system throughputs $X_c(\vec{n})$ for all classes c and all populations from no customers to the full population $\vec{N}$.
3. Create a high-level model consisting of centers $A+1$ through K, an FESC representing centers 1 through A, and customer population $\vec{N}$. The service rate of the FESC for class c when the customer population in its queue is $\vec{n}$ should be $X_c(\vec{n})$.
4. Evaluate this high-level model using the extension to MVA described in Chapter 20. The solution of this model is an approximation to the solution of the original K center network. System performance measures for all customer classes, and performance measures for centers $A+1$ through K, are obtained as the results of this solution. Performance measures for centers 1 through A can be computed by combining information from the solutions of the high- and low-level models. For instance, the average queue length at center K in a single class model with population N can be estimated as:

$$Q_K(N) = \sum_{n=1}^{N} \left[P[Q_{FESC}=n] \sum_{j=1}^{n} j \; P[Q_K=j \mid Q_{FESC}=n] \right]$$

where $P[Q_{FESC}=n]$ is the probability that the queue length at the FESC is n (obtained from the high-level model), and $P[Q_K=j \mid Q_{FESC}=n]$ is the probability that center K has queue length j given that there are n customers in the aggregate (obtained from the low-level model).

Algorithm 8.1 – A Hierarchical Decomposition Solution Technique for Separable Models

to acquire necessary parameters) to obtain an estimate of the performance of the system. The second is to create a non-separable model. A modification of the MVA solution algorithm that reflects the non-separable aspects of the model then is used to obtain approximate

performance measures. (Thus, we have an "exact" model but an approximate analysis technique.) Both of these approaches are used in Part III of this book. The final approach is to use a non-separable queueing network model and an analysis technique that yields the exact solution of the model. The price paid for this increased accuracy is that the solution requires a massive amount of computation.

In this section, we discuss the use of hierarchical modelling to decrease the cost of evaluating non-separable queueing network models. Our point of view is that we have determined that a non-separable queueing network model is required because of the need to represent a particular computer system characteristic, and are seeking a feasible means to evaluate this model. By judicious choices of aggregates, a large non-separable model can be replaced by a much smaller model, by substituting single FESCs for various subsystems of service centers. This (still non-separable) reduced model can be evaluated feasibly using one of the accurate but computationally expensive solution techniques for non-separable models. Thus, we have an approximate solution technique that allows explicit representation of very general features of computer systems and still is efficient enough to be practical.

In the next two subsections we examine two specific general solution techniques, one analytic and the other simulation.

8.5.1. Global Balance

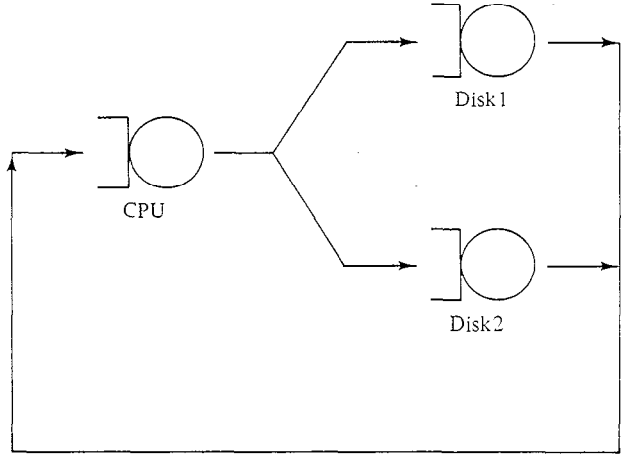
The general analytic technique used to evaluate closed, non-separable networks is called *global balance*. The global balance solution technique involves creating and solving the large sets of linear equations that describe the behavior of these models. This technique is impractically expensive in most cases because of the enormous number of equations and unknowns involved. Global balance requires one equation per state of the network, where a state is (roughly) a placement of customers at the service centers. A model with K centers and C classes therefore has at least:

$$\prod_{c=1}^{C} \begin{bmatrix} N_c + K - 1 \\ K - 1 \end{bmatrix}$$

equations and unknowns, where $\begin{bmatrix} n \\ p \end{bmatrix}$ denotes the number of ways of choosing p objects from n. Systems of equations of this size are unmanageable even for apparently modest K, C, and $\bar{N}$. For instance, a network with 6 service centers, 5 classes, and 5 customers in each class has more than 10^{12} states, and so cannot be solved directly using global balance.

The implication of the rapid growth in the size of the state space with the size of the model is that global balance can be applied only to very small models. Approximate solutions of large, general models can be obtained, however, by a combination of global balance and hierarchical decomposition. A large model is broken into pieces, each of which can be analyzed independently. These individual solutions then are combined into a single model using FESCs, and the solution of this much smaller model is obtained via global balance.

As an example, Figure 8.4 shows a model with three service centers (a CPU and two I/O devices) and two customer classes. Both I/O devices are queueing devices, while the CPU is scheduled with priority given to class A over class B. (An arriving class A customer goes into service immediately if there are no class A customers at the center, and queues behind those class A customers otherwise.) Because of the priority scheduling, the model is not separable, and thus cannot be evaluated using the MVA techniques of Chapter 7.



$N_A = 1$	$V_{A,CPU} = 16$	$V_{A,Disk1} = 15$	$V_{A,Disk2} = 0$
	$S_{A,CPU} = 15$	$S_{A,Disk1} = 20$	$S_{A,Disk2} = -$
	$D_{A,CPU} = 240$	$D_{A,Disk1} = 300$	$D_{A,Disk2} = 0$
$N_B = 2$	$V_{B,CPU} = 11$	$V_{B,Disk1} = 4$	$V_{B,Disk2} = 6$
	$S_{B,CPU} = 13$	$S_{B,Disk1} = 20$	$S_{B,Disk2} = 50$
	$D_{B,CPU} = 143$	$D_{B,Disk1} = 80$	$D_{B,Disk2} = 300$

Figure 8.4 – Global Balance Model

Recall that the service demand of class c at center k, $D_{c,k}$, is the product of the visit count, $V_{c,k}$, and the service requirement per visit, $S_{c,k}$. In separable models, we speak only of the $D_{c,k}$ because the performance measures are identical for all combinations of $V_{c,k}$ and $S_{c,k}$ that have the

same product $D_{c,k}$. In non-separable models, different combinations of $V_{c,k}$ and $S_{c,k}$ with the same product $D_{c,k}$ will in general yield different results. Thus, in order to specify the non-separable model in Figure 8.4, we have had to provide the $V_{c,k}$ and $S_{c,k}$. We assume that each job begins and ends service at the CPU, so for each class the CPU visit count is one greater than the sum of the disk visit counts. This information will be used only in obtaining the exact solution to the model; our hierarchical approximation will consider the model at the level of service demands.

This example is small enough that global balance could be applied directly. In general, however, this will not be the case. Yet, since priority scheduling has an important influence on the performance of the system, it is necessary to represent it in the model. We do so here by applying global balance to the smaller model created by replacing all centers other than the CPU with an FESC. (Other techniques for modelling priority scheduling are presented in Chapter 11.) The resulting two center model (the priority CPU and the FESC) then can be evaluated using global balance, and this solution used as an estimate for the performance measures of the system. The entire process is outlined below:

- *isolate the I/O subsystem* — A model consisting of only the I/O subsystem is created (see Figure 8.5). Each class has a service demand at the CPU of zero, and a service demand at each disk as indicated in Figure 8.4.

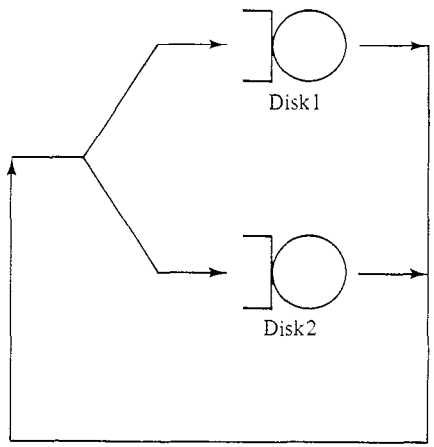


Figure 8.5 — Isolated I/O Subsystem Model

- *evaluate the low-level model* — The low-level model just created is evaluated for every population that it could contain in the full network. Since this submodel is separable, the standard MVA technique can be applied. The performance measures of interest are the population dependent throughputs for each class:

$\vec{n}$			
A	B	$X_A(\vec{n})$	$X_B(\vec{n})$
0	1	0	.00263
0	2	0	.00316
1	0	.00333	0
1	1	.00275	.00217
1	2	.00255	.00293

These give the rate at which customers leave the aggregate and return to the CPU for each customer population in the aggregate, and thus are the parameters required to form an FESC.

- *create the high-level model* — The high-level model (Figure 8.6) consists of the original CPU service center and an FESC representing the I/O subsystem. At the CPU, each class has the service demand indicated in Figure 8.4. The FESC has the population-dependent service rates shown in the preceding table (e.g., .00275 for class A and .00217 for class B when one customer of each class is present). Remember that the FESC is scheduled using composite queueing, so that all customer classes are in service simultaneously and independently. Thus, service rates of .00275 for class A and .00217 for class B mean that a class A customer will leave (on average) in 363.6 (= 1/.00275) time units and a class B customer in 460.8 (= 1/.00217).

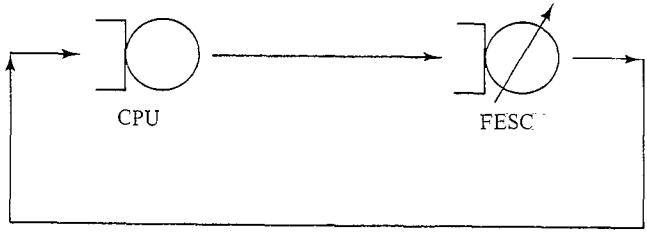


Figure 8.6 — The High-Level Model

- *evaluate the high-level model* — Since the high-level model contains a priority scheduled CPU service center, it cannot be solved using MVA (which pertains only to separable networks). However, the high-level model is small, and so can be solved by the global balance technique. We obtain:

$$X_A = .0016 \qquad X_B = .0020$$
$$Q_{A,CPU} = .396 \qquad Q_{B,CPU} = .838$$

The exact solution of the model of Figure 8.4, obtained by an expensive direct application of global balance, is:

$$X_A = .0016 \qquad X_B = .0020$$
$$Q_{A,CPU} = .373 \qquad Q_{B,CPU} = .790$$

Note that the performance measures obtained using the hierarchical approach are only approximations, although both the low- and high-level models were solved exactly. This is because the behavior of the I/O subsystem cannot be replicated exactly by the FESC, since information regarding the location of customers in the I/O subsystem is discarded.

The motivation for using an FESC in this example is that global balance can be applied to the resulting small high-level model, but (in more general cases) not to the original, large model. Use of the global balance technique was required because of the non-separable aspect of priority queueing in the model. In the following section we give a more detailed description of the global balance solution technique. The technique is described both in general terms, and more specifically as applied to the problem above. One should keep in mind that the global balance technique can be applied in many more situations than those involving priority scheduling. However, in all cases, the network to be solved must be quite small.

Details of Global Balance

The global balance solution technique can be used to compute the solutions of fairly general networks of queues. The technique is based on analyzing transitions of the system from one "state" to another.

We define a *state* of a *service center* in a queueing network model to be an ordering of customers in its queue. For example, the feasible states of a service center in a network with two class A customers and one class B customer are

$$(AAB) \quad (ABA) \quad (BAA) \quad (AA) \quad (AB) \quad (BA) \quad (A) \quad (B) \quad ( )$$

The state of a service center provides information about which customers are in service and which are waiting. In some cases the state description need not contain information about the ordering of customers in the queue. For instance, if the queue above were scheduled with priority to class A over class B, there would be no need to list the order of customers since it is certain that class A will be served first.

We define a *state* of a *queueing network* to be a composite of the states of all of its service centers. Intuitively, the state of a queueing network

contains all the information necessary to determine the behavior of the model at the moment.

We define the *state space* of a queueing network to be the set of feasible states. For instance, the state space of a model with two service centers and a single customer class of 3 customers is:

$$(3\ ;\ 0)\quad (2\ ;\ 1)\quad (1\ ;\ 2)\quad (0\ ;\ 3)$$

Here, the first number in each pair represents the number of customers at center one, and the second the number at center two. In general, the set of feasible states of a queueing model is determined by the number of customers of each class in the network, the service centers that each class visits, and the scheduling disciplines of the various centers.

We define a *state transition* to be the movement of the model from one of its states to another, caused by the movement of a customer within the model. For instance, if the model above were in state (3 ; 0), it would move to state (2 ; 1) when one of its customers completed service at center one and proceeded to center two. A common assumption made in analyzing queueing networks is that they exhibit *one step behavior:* each state transition involves the movement of exactly one customer. Thus, the network can move from state (3 ; 0) to state (2 ; 1), but not (directly) to state (1 ; 2). One step behavior is a reasonable assumption since it is very unlikely that any two jobs of the computer system can change locations at precisely the same time.

We define the *state transition rate* associated with a particular state transition to be the instantaneous rate at which that transition occurs, given that the network is in the starting state. For instance, if center one in the model above has a service time of 2 (a service rate of .5), and customers always alternate between centers 1 and 2, the rate associated with the transition from (3 ; 0) to (2 ; 1) is .5. In general, state transition rates depend on the service time of the moving customer at the center it departs, and the likelihood that a customer leaving this center proceeds immediately to another specific center. For single class models we have:

$$(n_1\ ;\ \ldots\ ;\ n_i+1\ ;\ \ldots\ ;\ n_j-1\ ;\ \ldots\ ;\ n_K) \rightarrow (n_1\ ;\ \ldots\ ;\ n_i\ ;\ \ldots\ ;\ n_j\ ;\ \ldots\ ;\ n_K)$$

with rate $\mu_i p_{i,j}$, where μ_i is the service rate of center i and $p_{i,j}$ is the proportion of time that a customer leaving center i proceeds directly to center j.

Given an arbitrary queueing network model, one can compute its state space, associated state transitions, and state transition rates from the model inputs. The solution of a model thus described can be obtained by making the *state space flow balance assumption* that the rate of flow of the network into any state must equal the rate of flow of the network out of that state. (This assumption is much like the flow balance assumption of

Chapter 3 applied to the network at the state space level.) The rate of flow out of a state S is the proportion of time spent in S multiplied by the sum of the state transition rates out of S. The rate of flow into a state S is the sum over every state of the network of the proportion of time spent in that state times the state transition rate from that state to S.

Finally, we define the *flow balance equations* to be the equations obtained by setting the total rate of flow into a state equal to the total rate of flow out of that state. The flow balance equations are a set of simultaneous linear equations in which the unknowns are the proportions of time spent in each possible network state. The global balance solution technique for queueing network models involves creating and solving these flow balance equations. Note that there is a single equation per state. Thus the complexity of global balance grows combinatorially with the size of the network, since the size of the state space does so.

As a particular example of the global balance technique we consider the solution of the high-level model of Figure 8.6.

- *create the state space* — Because the CPU uses priority scheduling, there is no need to include the order of customers in the queue there as part of the state description. Similarly, because the FESC uses composite queueing, the two customer classes act largely independently there and so queue ordering is not important. The model thus has six states. Using the notation (x;y) to indicate the state of the network with the CPU in state x and the FESC in state y, the state space of the model is:

  state1: $(ABB\ ;\ )$ state2: $(AB\ ;\ B)$ state3: $(BB\ ;\ A)$
  state4: $(A\ ;\ BB)$ state5: $(B\ ;\ AB)$ state6: $(\ ;\ ABB)$

- *calculate the state transition rates* — Each transition is caused by the movement of a customer from the CPU to the FESC or from the FESC to the CPU. The transition rate is equal to the rate at which this customer receives service at the origin center when in the origin state, multiplied by the proportion of time that this customer moves directly to the other (destination) center upon completion at the origin center.

  Because of the simple nature of the high-level model that we are considering, customers always move to the CPU upon completion at the FESC, and to the FESC upon completion at the CPU. Thus, $p_{A,CPU,FESC} = p_{B,CPU,FESC} = p_{A,FESC,CPU} = p_{B,FESC,CPU} = 1$. As a result, for example, the transition rate from state $(B;AB)$ to state $(AB;B)$, which involves the movement of a class A customer from the FESC to the CPU when one customer of each class is present at the FESC, is $.00275 \times 1 = .00275$. Figure 8.7 shows the state transition diagram for this model.

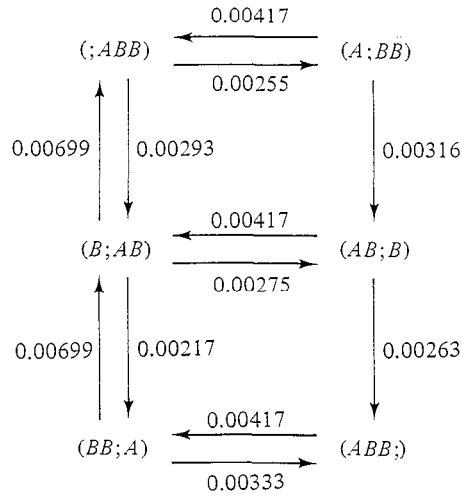


Figure 8.7 – The State Transition Diagram

- *create the flow balance equations* – The flow balance equations are obtained by setting flow in equal to flow out. The resulting set of equations do not determine a unique solution. Therefore, an arbitrary equation is discarded and replaced by an equation that ensures that the sum of the proportions of time spent in the states is one. In matrix notation, the balance equations for this example are:

$$\begin{vmatrix} -.00417 & .00263 & .00333 & 0 & 0 & 0 \\ 0 & -.00680 & 0 & .00316 & .00275 & 0 \\ .00417 & 0 & -.01032 & 0 & .00217 & 0 \\ 0 & 0 & 0 & -.00733 & 0 & .00255 \\ 0 & .00417 & .00699 & 0 & -.01191 & .00293 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{vmatrix} \begin{vmatrix} P(\text{state 1}) \\ P(\text{state 2}) \\ P(\text{state 3}) \\ P(\text{state 4}) \\ P(\text{state 5}) \\ P(\text{state 6}) \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{vmatrix}$$

- *solve the flow balance equations* – There are standard algorithms for solving sets of simultaneous linear equations. Gaussian elimination can be used on small systems. More sophisticated, iterative techniques may be required for larger models. The solution of the system of equations above gives the proportions of time spent in each state:

$$\begin{aligned} P(\text{state 1}) &= .161 \quad P(\text{state 4}) = .110 \\ P(\text{state 2}) &= .125 \quad P(\text{state 5}) = .183 \\ P(\text{state 3}) &= .104 \quad P(\text{state 6}) = .317 \end{aligned}$$

- *compute performance measures* – Performance measures may be calculated from the proportions of time spent in the various states. For instance, class A's *CPU* utilization is given by:

$$U_{A,CPU} = P(\text{state 1}) + P(\text{state 2}) + P(\text{state 4}) = .396$$

8.5.2. Hybrid Modelling

Hybrid modelling is a joint simulation/analytic solution technique that attempts to combine the best aspects of each. Simulation is used so that aspects of the computer system leading to non-separable models can be represented. Analytic techniques are used for efficiency.

To understand the relationship of hybrid modelling to the analytic techniques that are the primary concern of this book, we first must present a brief examination of the simulation approach to modelling. We have chosen to describe a particular type of simulation, that of probabilistic, event driven simulation. While other approaches are possible, event driven simulation is the most useful in computer system performance analysis.

Simulation techniques are experimental in nature. However, rather than running a physical experiment with real hardware and workload components (i.e., a benchmark experiment), the functional operation of the physical system is represented in software. The software maintains a simulation clock, which keeps track of the simulated elapsed time of the experiment. The software also keeps track of the state of each simulated physical device. States typically include information about which simulated jobs are in service or queued at each device, and information about the completion time of the operation in progress at each device. The software drives the simulation by selecting the event that should occur soonest, updating the simulation clock to the time of that event, and changing the state of the simulation to correspond to the occurrence of the event. This change of state might include the scheduling of new events at future simulation times. For instance, suppose that at simulation time 104.35 seconds, the next event that should occur is the completion of the job in service at the CPU at time 104.50 seconds. The simulation would advance the clock to 104.50 seconds, remove the job from the CPU queue, and enqueue that job at the device where it would next require service. It would also place a new job in service at the CPU (assuming that there were waiting jobs), pick a service time for that job according to some probability distribution that was an input parameter of the model (say 0.23 seconds), and schedule the departure of that job for some future simulation time (in this case at 104.73 seconds). The final task of the simulation driver is to record performance statistics about the simulation experiment. For instance, the driver might maintain a count of the total number of simulated seconds during which the simulated CPU was busy. At the end of the experiment, the ratio of that quantity to the final value of the simulation clock would be the estimate for CPU utilization.

It should be clear from this description that a simulation is capable of representing nearly arbitrary amounts of detail of the operation of the real

system. Of course, as more detail is incorporated, the size and expense of the simulation increase. Thus, to be useful, some amount of abstraction is required in forming the simulation model. For instance, a simulation model of a computer system might be identical to the queueing network models we have been examining (meaning that the input parameters of the simulation model and the queueing network model are the same). Alternatively, the simulation model might include more detail, such as a more accurate representation of a priority scheduling discipline used at the CPU. Finally, models with a large amount of detail (and very little abstraction) might include information about memory reference patterns (for use in determining page fault rates) or instruction mix (for use in determining effective CPU speed). Thus, simulation models are a superset of the queueing models with which we are concerned. Their advantage is their ability to incorporate detail. Their disadvantage is their expense: the computation required to obtain reliable performance estimates, the effort required to obtain the more detailed information needed to parameterize the more detailed models, and the effort required to gain insight into the critical parameters affecting performance in a model with a large number of inter-dependent parameters.

With this characterization of simulation in mind, we can proceed with the description of the basic hybrid modelling technique. Given a (non-separable) model of a system to be analyzed, isolate a subsystem (an aggregate of service centers) that can be solved conveniently in isolation. Create a flow equivalent service center to represent the submodel (by solving the submodel analytically to obtain the population dependent throughputs), and replace the subsystem by its FESC in the original model. Finally, solve this reduced model using simulation. Of course, it is possible to reverse the roles of simulation and queueing network modelling in this scheme (so that the low-level model is solved by simulation, and the high-level model analytically). This might be done, for instance, to model a complex I/O subsystem component of a large computer system, the remainder of which can be represented adequately as a separable queueing network.

In essence, this technique is identical to that of the previous subsection, with simulation substituted for global balance. Our motivation for proposing it also is the same: we have a powerful model solution technique (simulation) that we would like to employ, but the technique is too inefficient computationally for general use.

The inefficiency of simulation as a solution method is an effect of the statistical nature of the technique. Since simulation depends on observations of essentially random behavior sequences, many such sequences must be observed before we can have any confidence in the results (since any small number of sequences might be atypical). Thus, simulation is inherently expensive. This problem is compounded in cases where the

events being simulated happen at significantly differing rates. For example, consider a model in which the I/O subsystem is represented in detail, and from which we would like to obtain system throughput. Suppose that for each I/O request, we simulate individually the I/O path selection, cylinder seek, rotational latency, path reconnect, and data transfer times. Further, suppose that the effect of data transmission errors is represented by simulating each transferred byte (so that errors can be inserted). In this case we have events occurring at rates varying from relatively slow (job completions in the system) to relatively fast (byte transfers). As mentioned before, to obtain any statistical confidence in the results for system throughput, many job completions must be observed (say 1000, as an example). Suppose each job performs 100 I/O operations on average. This means 100,000 I/O operations must be simulated. Now suppose each I/O operation transfers 4,000 bytes of information. This implies the simulation of 400,000,000 byte transfers. Obviously such a simulation will require immense machine resources.

Hybrid modelling can be used to best advantage in situations like the above where there are large time scale differences in the rates at which various events take place. Typically, the subsystem containing the events occurring the most frequently is modelled analytically, and the load dependent throughputs obtained from the solutions are used to create an FESC. This FESC replaces the subsystem, and the resulting model is simulated. Activity in the subsystem therefore is represented by the arrival and departure of customers from the FESC, which must occur at the same rate as events in the remainder of the model (since that is where the customers come from). Thus, this model can be simulated (relatively) efficiently.

Consider using a model to evaluate the performance of various long term scheduling policies (memory admission policies). Let the model consist of service centers representing the significant hardware resources (CPU, disks, etc.), a memory queue, and three customer classes. One class represents CPU bound jobs, one I/O bound jobs, and one balanced jobs. The scheduling policies to be evaluated use information about the current memory resident job mix to select a waiting job from one of the three classes, in an attempt to maximize system throughput.

Because of the memory queue and complicated memory admission policies to be considered, this model is not separable and so cannot be solved analytically (although perhaps the technique of the previous section could be applied successfully). A pure simulation approach would be very expensive, if not infeasible, because of the time scale difference between the rate at which long term scheduling decisions must be made and the rate at which events occur within the central subsystem. Thus, a hybrid approach is recommended. The central subsystem (CPU and I/O subsystem) model is isolated, yielding a separable model. This model is

solved analytically for each feasible mix of customers of the three classes. Finally, a simulation of the memory admission policies is performed, with the time between job completions selected according to the rates of the FESC formed from the solutions of the central subsystem model solved previously. In essence, we use simulation to analyze a model consisting simply of the memory queue and an FESC representing the remainder of the computer system, with the parameters (service rates) of the FESC obtained by an analytic solution of the submodel the FESC replaces.

In an actual experiment with this technique applied to this problem, the maximum relative percentage difference between the hybrid technique and a simulation-only technique was 7%, while the simulation-only model took 56 times longer to execute. Given this combination of accuracy and efficiency, the hybrid technique is the approach of choice.

8.6. Summary

The key concept of this chapter is hierarchical decomposition, the process of splitting one model into a number of smaller submodels, each of which then can be analyzed in isolation. The solution of the original model is formed by combining the solutions of the submodels.

The submodels are combined using flow equivalent service centers. FESCs mimic the behavior of the submodels they represent by modelling the average output rates of these submodels as functions of their customer populations. Thus, FESCs are represented as load dependent service centers in the model.

The output rates of FESCs can be obtained in a number of ways, but by far the most important of these is the representation of the submodel as a queueing network model, which is solved by a single application of mean value analysis. Where this technique is applicable, it yields all the output rates for all populations of interest, and ensures that the FESC produced has analytically nice properties that allow efficient solutions of models that incorporate it. In some cases, however, this approach to solving the low-level model is not appropriate. (For instance, the parameter values of the low-level model might depend on the customer population. In this case the required load dependent rates cannot be obtained by a single application of MVA.) For these models, the load dependent rates used to parameterize the FESC generally will not lead to an efficiently analyzable higher-level model. We will deal with this problem in Part III of this book, when we use FESCs as tools in analyzing increasingly more sophisticated models of computer systems.

An important specific use of hierarchical modelling is the efficient approximate solution of non-separable queueing networks. There are two

important approaches to solving these models: global balance, and simulation. Both techniques can require excessive computation for all but very small models. Thus, to employ these techniques (and so to use the modelling constructs they allow) one must restrict the model size. Hierarchical modelling is useful in this respect because the large models that naturally arise in modelling computer systems can be reduced using flow equivalent service centers to models of manageable size.

In Part III of this book we examine a number of specific components of computer systems that must be represented in a performance model. In many cases we are confronted with characteristics of computer systems that cannot be modelled directly using separable networks. Hierarchical modelling and flow equivalent servers are the keys to successful models in many of these cases.

8.7. References

Flow equivalent service centers were shown to yield exact solutions for single class separable networks by Chandy et al. [1975]. Sauer and Chandy [1975] first presented their use as an approximation.

The global balance solution technique is a classical approach to the solution of Markovian systems (see [Cox & Miller 1965], for example). Sauer and Chandy [1981] present this material in the computer system modelling context.

The utility of the hybrid modelling approach of Section 8.5.2 was pointed out by Schwetman [1978] and Tolopka and Schwetman [1979]. For other case studies employing hybrid modelling, see [Browne et al. 1975] and [Lindzey & Browne 1979].

[Browne et al. 1975]
J.C. Browne, K.M. Chandy, R.M. Brown, T.W. Keller, D.F. Towsley, and C.W. Dissley. Hierarchical Techniques for Development of Realistic Models of Complex Computer Systems. *Proc. IEEE 63*,6 (June 1975), 966-975.

[Chandy et al. 1975]
K.M. Chandy, U. Herzog, and L.S. Woo. Parametric Analysis of Queueing Networks. *IBM Journal of Research and Development 19*,1 (January 1975), 50-57.

[Cox & Miller 1965]
D.R. Cox and H.D. Miller. *The Theory of Stochastic Processes.* Wiley, 1965.

[Lindzey & Browne 1979]
G.E. Lindzey, Jr. and J.C. Browne. Response Analysis of a Multi-Function System. *Proc. ACM SIGMETRICS Conference on Simulation, Measurement and Modeling of Computer Systems* (1979), 19-26.

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

[Sauer & Chandy 1981]
C.H. Sauer and K.M. Chandy. *Computer Systems Performance Modeling.* Prentice-Hall, 1981.

[Schwetman 1978]
H.D. Schwetman. Hybrid Simulation Models of Computer Systems. *CACM 21*,9 (September 1978), 718-723.

[Tolopka & Schwetman 1979]
S.J. Tolopka and H.D. Schwetman. Mix-Dependent Job Scheduling - An Application of Hybrid Simulation. *1979 National Computer Conference Proceedings, AFIPS Volume 48* (1979), AFIPS Press, 45-49.

8.8. Exercises

1. Modify the Fortran program of Chapter 18 to accommodate flow equivalent service centers. (The modifications required are described in Chapter 20.)
2. Use Algorithm 8.1 to evaluate a (separable) single class model consisting of a CPU center with service demand 10, and four disk centers with service demands 4, 3, 3, and 2. The customer class should be terminal type with 20 active users and 30 second think times. In applying the algorithm, treat the four disk centers as the aggregate, and the CPU center as the complementary network. Use the software created in answering Exercise 1 (extended to accommodate terminal classes) to analyze the high-level model that you construct. Compare the solution you obtain by applying hierarchical decomposition to that obtained by simply solving the full five-center network using MVA.
3. Use the global balance technique to solve the example model from Section 6.4.2.1. This exercise should illustrate dramatically the computational advantage of separable models (which can be solved using MVA) over general networks of queues (which require a global balance analysis to obtain the exact solution).

4. Figure 8.7 shows the state transition diagram for the model illustrated in Figure 8.6. There are two centers: a preemptive-priority-scheduled CPU, and an FESC representing the I/O subsystem. There are two classes: A, the high-priority class, with one customer, and B, the low-priority class, with two customers.
   a. Why is there no state $(BA;B)$?
   b. Why is there no transition from state $(BB;A)$ to state $(AB;B)$?
   c. Why is there no transition from state $(ABB;)$ to state $(AB;B)$?
   d. At what rate does class B depart the FESC when one class A and one class B customer are present there?