Chapter 6

Models with One Job Class

6.1. Introduction

In this chapter we examine *single class queueing network models.* Single class models are refinements of bounding models that provide *estimates* of performance measures, rather than simply bounds. For instance, instead of determining that the throughput of a certain system is between 1.1 and 2.0 jobs/minute (for a given population size), a single class model would provide an estimate of the actual throughput, such as 1.7 jobs/minute.

In single class models, the customers are assumed to be indistinguishable from one another. Although single class models always are simplifications, they nonetheless can be accurate representations of real systems. There are a number of situations in which a single class model might be used:

- *increased information* — The results of a bounding study might not provide sufficiently detailed information. Single class models are the next step in a progression of increasingly detailed models.
- *single workload of interest* — The computer system under consideration may be running only a single workload of significance to its performance. Therefore, it may not be necessary to represent explicitly the other workload components.
- *homogeneous workloads* — The various workload components of a computer system may have similar service demands. A reasonable modelling abstraction is to consider them all to belong to a single customer class.

Conversely, there are a number of situations in which it might be inappropriate to model a computer system workload by a single customer class. These situations typically arise either because distinct workload components exhibit markedly differing resource usage, or because the aim of the modelling study requires that inputs or outputs be specified in terms of the individual workload components rather than in terms of the aggregate workload. Typical instances of each are:

- *multiple distinct workloads* — On a system running both batch and timesharing workloads, the batch workload might be CPU bound while the timesharing workload is I/O bound. A queueing network model with a customer population consisting of a single class representing an "average" job might not provide accurate projections, since jobs in the actual system do not behave as though they were nearly indistinguishable.
- *class dependent model inputs* — In a mixed batch/timesharing system, the timesharing workload is expected to grow by 100% over the next 2 years, while the batch workload is expected to grow by only 10%. Since in a single class model there is only a single class of "average" customers, it is not possible to set the input parameters such that workload components exhibit differing growth rates. Thus, a single class model is not an appropriate representation.
- *class dependent model outputs* — In a batch environment running both production and development programs, projections about the time in system of each workload component, rather than just an estimate of "average" time in system, might be desired. Since there is only one class of customers in a single class model, outputs are given in terms of that class only, and it is difficult to interpret these measures in terms of the original classes of the system. Thus a multiple class model is required.

Systems having workloads with substantially differing characteristics, as exhibited by the examples above, may be modelled more reasonably by multiple class than by single class queueing networks. These more sophisticated models are discussed in Chapter 7.

The next two sections of this chapter deal with the practical application of single class queueing networks as models of computer systems. Section 6.2 discusses the use of the workload intensity parameter to mimic the job mix behavior of a computer system. Section 6.3 describes a number of case studies in which single class models have been employed.

This discussion of the practice of single class models is followed by a discussion of their theory. In Section 6.4 the algorithms required to evaluate the models are developed and illustrated with examples. Section 6.5 presents the theoretical underpinnings upon which the models rest.

6.2. Workload Representation

The workload representation of a single class queueing network model is given by two model inputs: the set of *service demands*, and the *workload intensity*. In using a single class model, one inherently makes the assumption that all jobs running in the system are sufficiently similar that their

differences do not have a major effect on system performance. Thus, calculating the set of service demands is fairly straightforward, as only a single set is required. (In contrast, with multiple class models one first must decide how many classes to represent, and then must calculate a distinct set of service demands for each class.)

Establishing the workload intensity has two aspects: selecting an appropriate workload type (transaction, batch, or terminal), and setting the appropriate workload intensity parameter(s) for that type. Selecting an appropriate workload type typically is straightforward, since the three workload types of queueing network models correspond directly to the three predominant workload types of computer systems. One technical distinction that arises is that between *open* models (those with transaction classes) and *closed* models (those with batch or terminal classes). Since the number of customers that may be in an open model at any time is unbounded, while the number of customers that may be in a closed model is bounded by the population of the closed class, the response times of open models tend to be larger than those of corresponding closed models with the same system throughput. This occurs because in open models the potential for extremely large queue lengths exists, while in closed models, because of the finite population, it does not. This difference usually is significant only when some device in the system is near saturation.

This brings us to the question of how to set the workload intensity parameter. In queueing network models, the workload intensity is a fixed quantity (an arrival rate, a population, or a population and a think time). In contrast, in a computer system the workload intensity may vary. Despite this discrepancy, queueing network models are useful in a wide variety of situations:

- *heavy load assumption* — It may be interesting to study the behavior of a system under the maximum possible load. By hypothesis, the load is sufficiently heavy that there always are jobs waiting to enter memory. Thus, when one job completes and releases memory, it immediately is replaced by another job. The workload therefore is represented as a batch class with a constant number of customers equal to the maximum multiprogramming level of the system.
- *non-integer workload intensity* — The measurement data for a system might show that the average multiprogramming level (or active number of terminal users) is not an integer. Some algorithms for evaluating queueing network models allow non-integer customer populations. Other algorithms do not. For the latter, the model can

be evaluated for the neighboring integer workload intensity values and the non-integer solution obtained by interpolation. For instance, if the measured multiprogramming level were 4.5, the solutions of the model with batch populations of 4 and 5 could be computed, and their average taken as the projection for 4.5 customers.

- *workload intensity distribution* — Measurement data might provide a distribution of observed workload intensities, e.g., proportions of time $P[N=n]$ that there were n active terminal users on the system. This distribution could be used to weight the solutions obtained for a model with each observed number of users. Table 6.1 gives an example.

n	$P[N=n]$	$U_{CPU}(n)$	$X(n)$	$R(n)$
0	.1	0	0	0
1	.2	.032	.0525	.787
2	.3	.062	.1031	1.546
3	.3	.092	.1515	2.273
4	.1	.119	.1974	2.961

$$U_{CPU} = \sum_{n=1}^{4} P[N=n]\,U_{CPU}(n) = .0645$$

$$R = \sum_{n=1}^{4} \left[\frac{X(n)P[N=n]}{\sum_{j=1}^{4} X(j)P[N=j]} \; R(n) \right] = 2.492$$

Table 6.1 — Use of Distributional Information

- *sizing studies* — Because the solutions of single class models can be obtained extremely quickly, it is feasible to evaluate a model for a large number of workload intensities. Thus, questions such as "What is the maximum transaction arrival rate that can be supported with average response time below 3 seconds?" can be answered by varying the arrival rate of a model (e.g., setting $\lambda = 1, 2, \ldots$) and observing the reported response times.
- *robustness studies* — Similarly, since it often is the case that workload growth cannot be forecast accurately, it generally is useful to evaluate a model for a range of workload intensities surrounding the expected one. This allows the analyst to assess the impact on projected performance of a growth in the workload that exceeds expectations.

6.3. Case Studies

Three applications of single class queueing network models are described in this section. The first is a classic study in which an extremely simple model gave surprisingly accurate performance projections. The second is an application in which the effects of modifying certain hardware and software characteristics were investigated. The third illustrates a recent use of a single class model for capacity planning.

6.3.1. A Model of an Interactive System

We first consider what may be the earliest application of queueing network modelling to computer systems. We include this study despite its age (it was performed in 1965) because of its historical interest and because it demonstrates vividly that extremely simple models can be accurate predictors of performance.

The system under study was an IBM 7094 running the Compatible Time-Sharing System (CTSS). CTSS was an experimental interactive system based on swapping. Only a single user could be "active" at a time. The entire system — CPU, disks, and memory — was "time-sliced" among users as a unit.

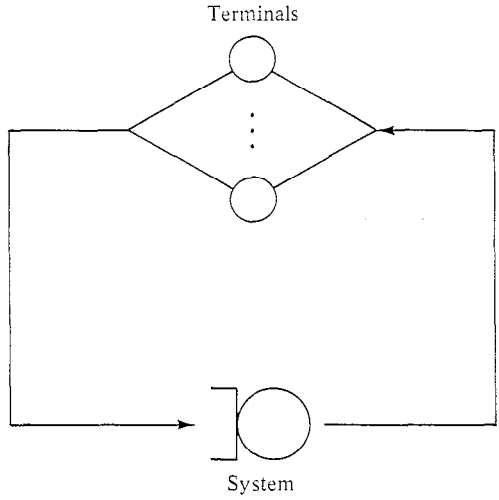


Figure 6.1 — Interactive System Model

The purpose of the study was to investigate the response time behavior of the system as a function of the number of users. To do so, the model of Figure 6.1 was constructed. It contains a terminal workload,

representing the user population, and a single service center representing the system (CPU and disks). This single service center representation is sufficient because, with only one user active at a time, there can be no overlap in processing at the CPU and the disks (individual users on this system did not exploit this capability). Thus, in terms of average response time, it does not matter (in the model) whether a user spends time at the CPU or the disks, but simply that the appropriate amount of time transpires.

Notice that by using a single service center to represent the system, we have solved a simple memory constraint problem. Had the model contained separate CPU and disk service centers, it would have been less accurate because it would have allowed customers to be processing at both simultaneously, while in the actual system this was not possible. This technique of collapsing a number of service centers into a single service center to represent memory constraints can be extended in quite powerful ways, as will be explained in Chapter 9.

The model was parameterized from measurements taken during system use, which provided average think time, average CPU and disk processing times, and average memory requirement. The service demand at the system service center was set equal to the sum of the measured processing times and the disk service required for swapping a job of average size. The number of customers in the model then was varied, and response time estimates for each population were obtained. Figure 6.2 compares the model projections with measured response times.

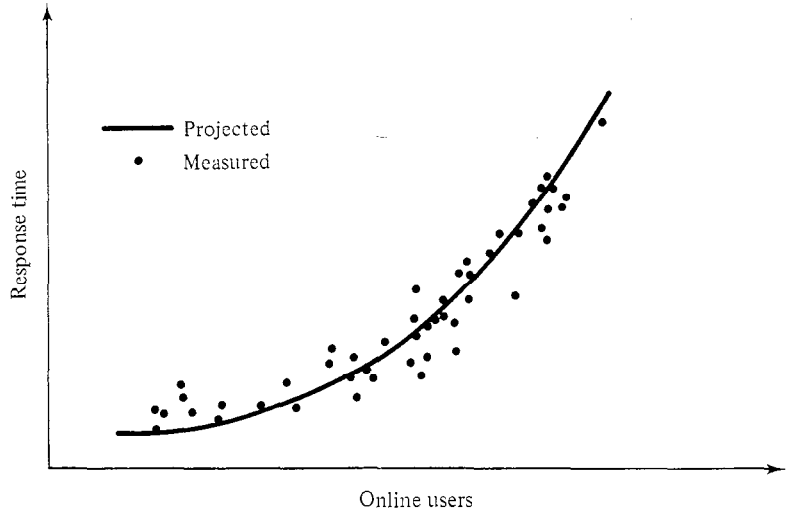


Figure 6.2 — Measured and Projected Response Times

6.3.2. A Model with Modification Analysis

In this case study a single class model was used to evaluate the benefits of several proposed changes to a hardware and software configuration. The system under consideration was an IBM System/360 Model 65J with three channels. Channels one and two were connected to 8 and 16 IBM 2314-technology disks, respectively. Channel three was connected to a drum, which was used exclusively by the operating system. Because the use of this drum was overlapped entirely with the processing of user jobs, it was omitted from the model. (Customers in the model represent user jobs, which never visited the drum.)

The model of this system is shown in Figure 6.3. It is parameterized by specifying service demands for the CPU, disks, and channels, as well as the workload type and intensity. The model differs from our "standard" model (cf. Section 4.5) because of the inclusion of service centers representing the channels. In general, a model of this sort can lead to significant error (as will be explained shortly). However, because of the characteristics of this system, good accuracy was obtained.

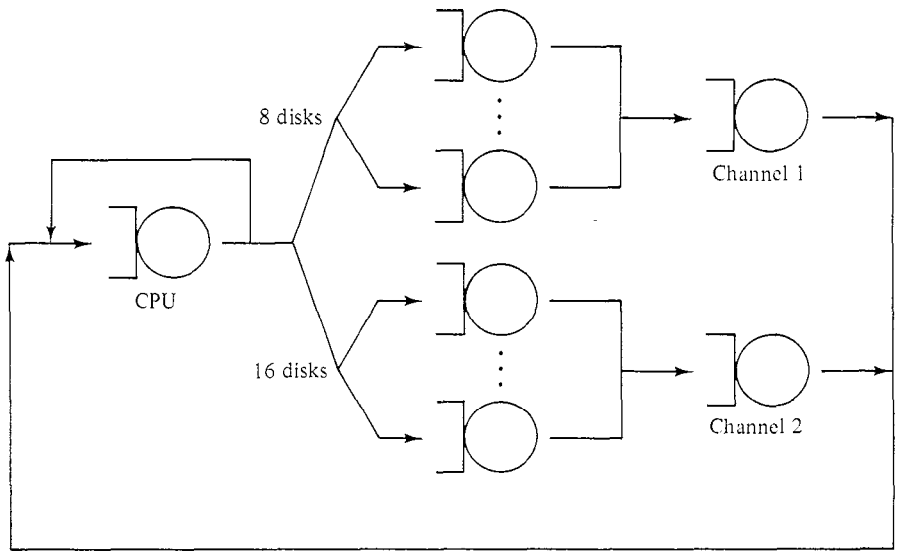


Figure 6.3 – System Model

The base CPU service requirement per job was estimated by dividing the total CPU busy time (both system and user time) over the measurement interval by the number of jobs that completed in the interval. Thus, system CPU overhead (such as that required to handle CPU scheduling and user I/O) was allocated equally among all jobs.

Parameterizing the I/O subsystem (the disks and channels) was more complicated. The disk technology of the system required that both the disk and the channel be held during rotational latency (the period during which the data is rotating to the read/write heads of the disk) and data transfer, while seeks could proceed at each disk independently of its channel. In the model, the channel service demands were set to the sum of the average latency and data transfer times, while the disk service demands were set to the average seek time. Thus, all components of I/O service time were represented exactly once. (If all three components of service were represented at the disks, customers in the model would experience latency and transfer service twice, and projected performance measures would be seriously in error.)

There is a danger in representing multiple component I/O subsystems in this manner. Unlike the actual system, no customer in the model ever holds both a disk and a channel simultaneously. Thus, there is the potential for artificial parallelism in the model, since a disk center that logically is being used for the latency and transfer portion of one job's service might be used at the same time to seek by another job. Accounting for this inaccuracy in general is a difficult problem. (Chapter 10 discusses I/O modelling in more detail.) However, in the case of this particular system, the effect of the potential parallelism in the model was negligible because the utilizations of the disk devices were fairly well balanced, and the total number of disks was much larger than the average multiprogramming level. Thus, the probability that a customer would require service from a disk already in use by another customer was small, and consequently so was the amount of artificial parallelism.

Measurement of the system showed that the average multiprogramming level varied significantly during the measurement interval. To account for this variability, the model was evaluated once for each observed multiprogramming level. Performance projections were obtained by weighting the distinct solutions by the percentage of time the corresponding multiprogramming levels were observed in the system.

The purpose of the modelling study was to evaluate the effects of the following proposed changes to the system:

- Replace eight of the 2314-technology disks on one channel with six IBM 3330 disks. The effect of this change was reflected in the model by altering the service demands of the affected channel and disk service centers, since 3330s seek and transfer data faster than 2314s, and also have *rotational position sensing (RPS)* capability, which allows the disk to disconnect from the channel during rotational latency, reconnecting only when the required sector is about to come under the read/write heads.

- Replace *extended core storage (ECS)* with faster memory. This would result in an effectively faster CPU, since the processing rate was limited by the memory access time. As most of the programs executed out of ECS were system routines, the effect of this change was reflected in the model by reducing the portion of the CPU service demand corresponding to supervisor state (system) processing.
- Implement an operating system improvement. This improvement was expected to reduce overhead by 8%. Thus this change was reflected in the model by decreasing the portion of the CPU service demand corresponding to operating system processing.

The model was parameterized to reflect various combinations of the proposed system improvements, and the effect on user (problem state) CPU utilization was noted. (The use of U_{CPU} as the performance metric is an odd aspect of this study, since U_{CPU} can be made to increase simply by slowing down the processor. More typical metrics are system throughput and system response time.) The operating system improvement alone was projected to yield a 5% increase in U_{CPU}. In conjunction with the ECS replacement, the gain was projected to be 25%. When the operating system improvement was combined with the disk upgrade, a similar 25% gain was projected. This pair of modifications actually was implemented; subsequent measurements showed that U_{CPU} had increased by about 20%, even though the basic CPU service demand had diminished due to an unanticipated change in the workload. Thus, the model provided a close projection of true system behavior.

This example shows that quite simple models can be used to answer performance questions of interest. It is important to notice how little of the detail of the computer system is represented in the model; only those aspects of the system that were crucial to performance and under consideration for modification were represented. For example, there is no explicit representation of memory in the model. This simplicity is a great advantage of queueing network models.

6.3.3. Capacity Planning

The purpose of this study was to evaluate the impact on response time of an anticipated 3% quarterly growth in the volume of the current workload. The system was an Amdahl 470 with 8 MB of main store, 16 channels, and 40 disks. The system was running IBM's MVS operating system and IMS database system, running a transaction processing workload. IMS was supporting five *message processing regions*: areas of main memory allocated and scheduled by IMS, each of which can accommodate one user request. If more than five requests were outstanding, the remainder queued for an available region.

Many different transaction types existed in the system. However, they were increasing in volume at about the same rate, so a single class model was sufficient to investigate the performance question of interest. (If various transaction types had been growing at differing rates, a multiple class model would have been required.) The model of the system is shown in Figure 6.4. It contains a single transaction workload, representing the aggregate of all the transaction types in the system, a memory queue, reflecting the fact that only five message processing regions were available, a CPU service center, and 40 disk service centers.

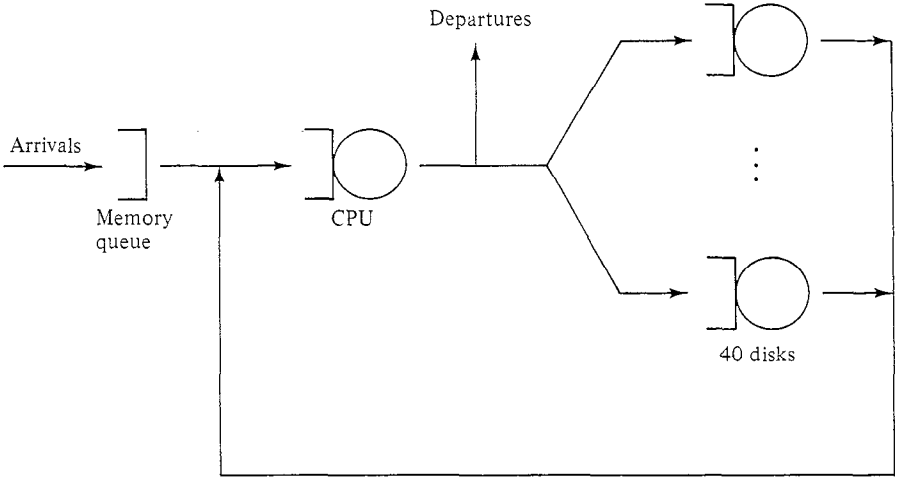


Figure 6.4 – System Model

Because this model contains a memory queue, it is not separable, and so cannot be evaluated directly by the techniques to be introduced later in this chapter. In Chapter 9 we discuss general methods for evaluating models of this type. For now, it is sufficient to observe that the solution of an open model with a saturated memory queue is roughly equivalent to the solution of a corresponding closed model in which the open class of customers has been replaced by a closed class with multiprogramming level equal to maximum possible number of simultaneously active jobs. This model is separable, so can be evaluated easily.

Parameters for the model were obtained from information gathered by software monitors:

- The arrival rate of customers was set equal to the measured transaction arrival rate.

- The service demand at the CPU was set equal to:

$$D_{CPU} = U_{CPU} \, T / C$$

where U_{CPU} was the measured *CPU* utilization, T was the length of the measurement interval, and C was the number of transactions that completed during the interval.

- The service demand at each disk k was set equal to:

$$D_k = U_k \, T / C$$

Notice that because of the way the service demands were calculated, both overhead and inherent service requirements were included. In the case of the CPU, this means that both user and system processing time were accounted for. In the case of the disks, this means that seek, rotational latency, data transfer, and any time lost because of I/O path contention were included. This approach to accounting for overhead can be quite useful when it is anticipated that the ratio of overhead to useful processing time will be relatively insensitive to the proposed modifications being investigated. The advantage of this approach is the simple way in which service demands can be computed. (For example, we do not need to determine the duration of each component of disk service time.) The disadvantage is that anticipated changes in the ratios of overhead to inherent service times cannot be modelled without more detailed information. For the modifications considered in this study, it was not felt that this was a significant drawback.

Having set the parameters, the model was evaluated to obtain response time projections. Figure 6.5 graphs projected response time against year for four different memory sizes: the existing configuration, adequate to support five message processing regions, and expanded configurations supporting six, seven, and eight message processing regions. On the basis of this study, it was concluded that, with the addition of memory, the system would be adequate for at least two years.

6.4. Solution Techniques

The solution of a queueing network model is a set of performance measures that describe the time averaged (or long term) behavior of the model. Computing these measures for general networks of queues is quite expensive and complicated. However, for separable queueing network models, solutions can be obtained simply.

The specific procedures followed to analyze separable queueing networks differ for open and closed models. We consider each in turn.

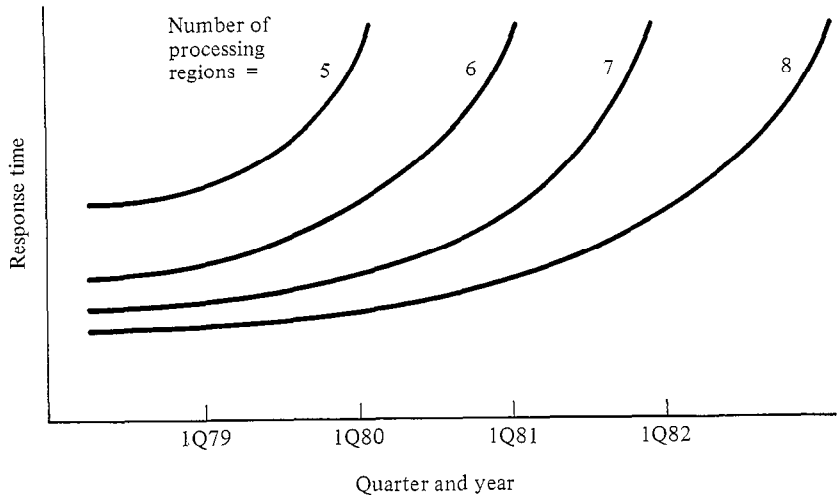


Figure 6.5 – Projected Response Times

6.4.1. Open Model Solution Technique

For open models (those with transaction workloads), one of the key output measures, system throughput, is given as an input. Because of this, the solution technique for these models is especially simple. We list here the formulae that apply for each performance measure of interest.

- *processing capacity*

  The processing capacity of an open model, λ_{sat}, is the arrival rate at which it saturates. This is given by:

$$\lambda_{sat} = \frac{1}{\max_k D_k} = \frac{1}{D_{max}}$$

  In the derivations that follow, we assume that $\lambda < \lambda_{sat}$.

- *throughput*

  By the forced flow law, if λ customers/second enter the network, then the system output rate must also be λ customers/second. Similarly, if each customer requires on average V_k visits to device k, the throughput at device k must be λV_k visits/second. Thus:

$$X_k(\lambda) = \lambda V_k$$

- *utilization*

  By the utilization law, device utilization is equal to throughput multiplied by service time. Thus:

$$U_k(\lambda) = X_k(\lambda) S_k = \lambda D_k$$

  (In the case of delay centers, the utilization must be interpreted as the average number of customers present.)

- *residence time*

  The residence time at center k, $R_k(\lambda)$, is the total time spent at that center by a customer, both queueing and receiving service. For service centers of delay type, there is no queueing component, so $R_k(\lambda)$ is simply the service time multiplied by the number of visits:

$$R_k(\lambda) = V_k S_k = D_k \qquad \text{(delay centers)}$$

  For queueing centers, R_k is the sum of the total time spent in service and the total time spent waiting for other customers to complete service. The former component is $V_k S_k$. The latter component is the time spent waiting for customers already in the queue when a customer arrives. Letting $A_k(\lambda)$ designate the average number of customers in queue as seen by an arriving customer, the queueing component is $V_k\left[A_k(\lambda)S_k\right]$. (By assumption, to be discussed in Section 6.5, the expected time until completion of the job in service when a new job arrives is equal to the service time of the job.) Thus, for queueing centers the residence time is given by:

$$\begin{aligned} R_k(\lambda) &= V_k\left[S_k + S_k A_k(\lambda)\right] \\ &= D_k\left[1 + A_k(\lambda)\right] \end{aligned}$$

  An implication of the assumptions made in constructing separable networks is that the queue length seen upon arrival at center k, $A_k(\lambda)$, is equal to the time averaged queue length $Q_k(\lambda)$, giving:

$$R_k(\lambda) = D_k\left[1+Q_k(\lambda)\right]$$

  which, using Little's law to re-express Q_k, is:

$$\begin{aligned} R_k(\lambda) &= D_k\left[1+\lambda R_k(\lambda)\right] \\ &= \frac{D_k}{1-U_k(\lambda)} \qquad \text{(queueing centers)} \end{aligned}$$

  This equation exhibits the intuitively appealing property that as $U_k(\lambda)\to 0$, $R_k(\lambda)\to D_k$, and as $U_k(\lambda)\to 1$, $R_k(\lambda)\to\infty$.

- *queue length*

  By Little's law: $Q_k(\lambda) = \lambda R_k(\lambda)$

$$= \begin{cases} U_k & \text{(delay centers)} \\ \dfrac{U_k(\lambda)}{1-U_k(\lambda)} & \text{(queueing centers)} \end{cases}$$

- *system response time*

  System response time is the sum of the residence times at all service centers:

$$R(\lambda) = \sum_{k=1}^{K} R_k(\lambda)$$

- *average number in system*

  The average number in system can be calculated using Little's law, or by summing the queue lengths at all centers:

$$Q(\lambda) = \lambda R(\lambda) = \sum_{k=1}^{K} Q_k(\lambda)$$

These formulae are summarized as Algorithm 6.1.

processing capacity : $\lambda_{sat} = 1 / D_{max}$

throughput : $X(\lambda) = \lambda$

utilization : $U_k(\lambda) = \lambda D_k$

residence time : $R_k(\lambda) = \begin{cases} D_k & \text{(delay centers)} \\ \dfrac{D_k}{1 - U_k(\lambda)} & \text{(queueing centers)} \end{cases}$

queue length : $Q_k(\lambda) = \lambda R_k(\lambda)$

$$= \begin{cases} U_k(\lambda) & \text{(delay centers)} \\ \dfrac{U_k(\lambda)}{1 - U_k(\lambda)} & \text{(queueing centers)} \end{cases}$$

system response time : $R(\lambda) = \sum_{k=1}^{K} R_k(\lambda)$

average number in system : $Q(\lambda) = \lambda R(\lambda) = \sum_{k=1}^{K} Q_k(\lambda)$

Algorithm 6.1 – Open Model Solution Technique

Open Model Example

Figure 6.6 shows a simple open model with three service centers, and illustrates the calculation of various performance measures. (All times are in seconds.)

6.4.2. Closed Model Solution Techniques

The technique we use to evaluate closed queueing networks (those with terminal or batch classes) is known as *mean value analysis* (*MVA*). It is based on three equations:

- *Little's law applied to the queueing network as a whole* :

$$X(N) = \frac{N}{Z + \sum_{k=1}^{K} R_k(N)} \tag{6.1}$$

where $X(N)$ is the system throughput and $R_k(N)$ the residence time at center k, when there are N customers in the network. (As usual, if the customer class is batch type, we take $Z = 0$.) Note that system throughput can be computed from input parameter data if the device residence times $R_k(N)$ are known.

- *Little's law applied to the service centers individually* :

$$Q_k(N) = X(N)R_k(N) \tag{6.2}$$

Once again, the residence times must be known before Little's law can be applied to compute queue lengths.

- *The service center residence time equations* :

$$R_k(N) = \begin{cases} D_k & \text{(delay centers)} \\ D_k\left[1 + A_k(N)\right] & \text{(queueing centers)} \end{cases} \tag{6.3}$$

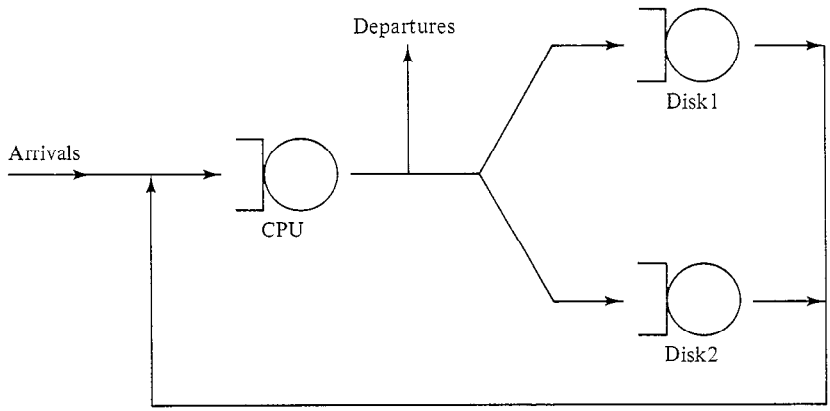
where $A_k(N)$ is the average number of customers seen at center k when a new customer arrives.

Note that, as with open networks, the key to computing performance measures for closed networks is the set of $A_k(N)$. If these were known, the $R_k(N)$ could be computed, followed by $X(N)$ and the $Q_k(N)$. In the case of open networks we were able to substitute the time averaged queue lengths, $Q_k(N)$, for the arrival instant queue lengths, $A_k(N)$. In the case of closed networks, this substitution is not possible. To see that $A_k(N)$ does not equal $Q_k(N)$ in closed networks, consider the network consisting of two queueing service centers and a single customer with a service

Model Inputs:

$$V_{CPU} = 121 \qquad V_{Disk1} = 70 \qquad V_{Disk2} = 50$$
$$S_{CPU} = .005 \qquad S_{Disk1} = .030 \qquad S_{Disk2} = .027$$
$$D_{CPU} = 0.605 \qquad D_{Disk1} = 2.1 \qquad D_{Disk2} = 1.35$$
$$\lambda = 0.3 \text{ jobs/sec.}$$

Model Structure:



Selected Model Outputs:

$$\lambda_{sat} = \frac{1}{D_{max}} = \frac{1}{2.1} = .476 \text{ jobs/sec.}$$

$$X_{CPU}(.3) = \lambda V_{CPU} = (.3)(121) = 36.3 \text{ visits/sec.}$$

$$U_{CPU}(.3) = \lambda D_{CPU} = (.3)(.605) = .182$$

$$R_{CPU}(.3) = \frac{D_{CPU}}{1 - U_{CPU}(.3)} = \frac{.605}{.818} = .740 \text{ secs.}$$

$$Q_{CPU}(.3) = \frac{U_{CPU}(.3)}{1 - U_{CPU}(.3)} = \frac{.182}{.818} = .222 \text{ jobs}$$

$$R(.3) = R_{CPU}(.3) + R_{Disk1}(.3) + R_{Disk2}(.3)$$
$$= .740 + 5.676 + 2.269 = 8.685 \text{ secs.}$$

$$Q(.3) = \lambda R(\lambda) = (.3)(8.685) = 2.606 \text{ jobs}$$

Figure 6.6 – Open Model Example

demand of 1 second at each center. Since there is only one customer, the time averaged queue lengths at the service centers are simply their utilizations, so $Q_1(1) = Q_2(1) = 1/2$. However, the arrival instant queue lengths $A_1(1)$ and $A_2(1)$ both are zero, because with a single customer in the network no customers could possibly be in queue ahead of an arriving customer. In general, the key distinction is that the arrival instant queue lengths are computed conditioned on the fact that some customer is arriving to the center (and so cannot itself be in the queue there), while the time averaged queue lengths are computed over randomly selected moments (so all customers potentially could be in the queue).

As mentioned above, evaluating a model requires that we first compute the $A_k(N)$. There are two basic techniques, exact and approximate. We emphasize that this distinction refers to how the solution relates to the model, rather than to the computer system itself. The accuracy of the solution relative to the performance of the computer system depends primarily on the accuracy of the parameterization of the model, and not on which of the two solution techniques is chosen.

We next examine each of the two solution methods, beginning with the exact technique.

6.4.2.1. Exact Solution Technique

The exact MVA solution technique is important for two reasons:

- It is the basis from which the approximate technique is derived.
- There are no known bounds on the inaccuracy of the approximate technique. While typically it is accurate to within a few percent relative to the true solution, it cannot be guaranteed that in any particular situation the results will not be worse.

The exact solution technique involves computing the arrival instant queue lengths $A_k(N)$ exactly, then applying equations (6.1)-(6.3). The characteristic of closed, separable networks that makes them amenable to this approach is that the $A_k(N)$ have a particularly simple form:

$$A_k(N) = Q_k(N-1) \tag{6.4}$$

In other words, the queue length seen at arrival to a queue when there are N customers in the network is equal to the time averaged queue length there with one less customer in the network. This equation has an intuitive justification. At the moment a customer arrives at a center, it is certain that this customer itself is not already in queue there. Thus, there are only $N-1$ other customers that could possibly interfere with the new arrival. The number of these that actually are in queue, on average, is simply the average number there when only those $N-1$ customers are in the network.

The exact MVA solution technique, shown as Algorithm 6.2, involves the iterative application of equations (6.1)-(6.4). These equations allow us to calculate the system throughput, device residence times, and time averaged device queue lengths when there are n customers in the network, given the time averaged device queue lengths with $n-1$ customers. The iteration begins with the observation that all queue lengths are zero with zero customers in the network. From that trivial solution, equations (6.1)-(6.4) can be used to compute the solution for one customer in the network. Since the time averaged queue lengths with one customer in the network are equal to the arrival instant queue lengths with two customers in the network, the solution obtained for a population of one can be used to compute the solution with a population of two. Successive applications of the equations compute solutions for populations 3 , 4 , ... , N.

for $k \leftarrow 1$ to K do $Q_k \leftarrow 0$
for $n \leftarrow 1$ to N do
begin

$$\text{for } k \leftarrow 1 \text{ to } K \text{ do } R_k \leftarrow \begin{cases} D_k & \text{(delay centers)} \\ D_k(1 + Q_k) & \text{(queueing centers)} \end{cases}$$

$$X \leftarrow \frac{n}{Z + \sum_{k=1}^{K} R_k}$$

for $k \leftarrow 1$ to K do $Q_k \leftarrow XR_k$
end

Algorithm 6.2 – Exact MVA Solution Technique

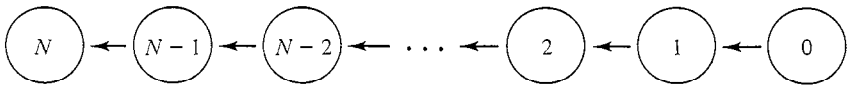


Figure 6.7 – Single Class Solution Population Precedence

Figure 6.7 illustrates the precedence relations of the solutions required to apply the exact MVA solution technique. As just described, the solution of a closed model with N customers requires the solution with $N-1$ customers, which requires the solution with $N-2$ customers, etc. Thus,

the full solution requires N applications of equations (6.1)-(6.4). Since each of the N applications of the equations requires looping (several times) over the K service centers, the computational expense of the solution grows as the product of N with K. The space requirement, in contrast, is about K locations, since the performance measures for the network with n customers can be discarded once they have been used to calculate the performance measures for $n+1$ customers. Note that all solutions between 1 customer and N customers are computed as by-products of the N customer solution. Thus, there is no additional expense involved in obtaining these intermediate solutions (although of course some additional space is required if all of them are to be retained). This is an important characteristic of the solution technique that will be exploited in Chapter 8 when we discuss flow equivalent service centers.

When Algorithm 6.2 terminates, the values of R_k, X, and Q_k (all for population N) are available immediately. Other model outputs are obtained by using Little's law. Here is a summary:

system throughput:	X
system response time:	$N/X - Z$
average number in system:	$N - XZ$
device k throughput:	XV_k
device k utilization:	XD_k
device k queue length:	Q_k
device k residence time:	R_k

Closed Model Example (Exact Solution)

Table 6.2 shows the computation of the solution of the network of Figure 6.6 with the transaction class replaced by a terminal class. There are three centers, with service demands $D_{CPU} = .605$ seconds, $D_{Disk1} = 2.1$ seconds, and $D_{Disk2} = 1.35$ seconds. The terminal class has three customers ($N=3$) and average think time of 15 seconds ($Z=15$). The algorithm begins with the known solution for the network with zero customers, and calculates the $R_k(n)$, $X(n)$, and $Q_k(n)$ for each successively larger population n, up to three.

In studying Table 6.2, note that the sum of the queue lengths at the three centers does not equal the customer population. This is the case because we are dealing with a class of terminal type, and some of the customers are "thinking". (Algorithm 6.2 accounts for this by the inclusion of the think time, Z, in one of its equations.) We can calculate the average number of "thinking" customers by subtracting the average number in system, $Q = N - XZ$, from the total customer population, N, yielding XZ (which equals zero for a batch class).

	k	$n=0$	$n=1$	$n=2$	$n=3$
R_k	*CPU*	-	.605	.624	.644
	*Disk*1	-	2.1	2.331	2.605
	*Disk*2	-	1.35	1.446	1.551
X		-	.0525	.1031	.1515
Q_k	*CPU*	0	.0318	.0643	.0976
	*Disk*1	0	.1102	.2403	.3947
	*Disk*2	0	.0708	.1490	.2350

Table 6.2 – Exact MVA Computation

Model outputs can be computed from the results for $N=3$:

$$
\begin{aligned}
X(3) &= .152 \\
R(3) &= 3/.152 - 15.0 = 4.74 \\
Q(3) &= N - X(3)Z = 3 - (.152)(15) = .72 \\
X_{CPU}(3) &= X(3)V_{CPU} = (.152)(121) = 18.39 \\
U_{CPU}(3) &= X(3)D_{CPU} = (.152)(.605) = .092 \\
Q_{CPU}(3) &= .098 \\
R_{CPU}(3) &= .64
\end{aligned}
$$

6.4.2.2. Approximate Solution Technique

The key to the exact MVA solution technique is equation (6.4), which computes the arrival instant queue length for population n based on the time averaged queue length with population $n-1$. The nature of the algorithm is a direct consequence of this relationship.

By replacing equation (6.4) with an approximation:

$$A_k(N) \approx h\left[Q_k(N)\right]$$

for some suitable function h, a more efficient, iterative algorithm can be obtained. (The function h actually might depend on values other than $Q_k(N)$. For instance, the approximation we will propose shortly also depends on N. However, we use this notation for simplicity, and to suggest that the key requirement is the value of $Q_k(N)$.) The accuracy of the algorithm depends, of course, on the accuracy of the function h that is used. (A particular choice for h will be presented shortly.)

This general approach is outlined in Algorithm 6.3. It is seen easily that the time and space requirements of this algorithm depend on the number of centers but are independent of the customer population of the network being evaluated (except indirectly; the number of iterations required for convergence may be affected by the population). This can be

a substantial improvement over the exact MVA technique, which requires time proportional to the product of the number of centers and the number of customers.

1. Initialize: $Q_k(N) \leftarrow \frac{N}{K}$ for all centers k.
2. Approximate: $A_k(N) \leftarrow h\left[Q_k(N)\right]$ for all centers k.
   (The choice of an appropriate function h is discussed in the text.)
3. Use equations (6.3), (6.1), and (6.2) in succession to compute a new set of $Q_k(N)$.
4. If the $Q_k(N)$ resulting from Step 3 do not agree to within some tolerance (e.g., 0.1%) with those used as inputs in Step 2, return to Step 2 using the new $Q_k(N)$.

Algorithm 6.3 – Approximate MVA Solution Technique

Crucial to this faster solution technique is the function h. Unfortunately, no function h is known that is exact for all separable networks. Instead, an approximation must be used. A particularly simple and reasonably accurate approximation is:

$$\begin{aligned} A_k(N) &= Q_k(N-1) \\ &\approx h\left[Q_k(N)\right] \\ &\equiv \frac{N-1}{N} Q_k(N) \end{aligned} \tag{6.5}$$

Equation (6.5) estimates the arrival instant queue length by approximating its exact value, the queue length with one fewer customer. This approximation is based on the assumption that the ratios $\frac{Q_k(N)}{N}$ and $\frac{Q_k(N-1)}{N-1}$ are equal for all k, i.e., that the amount that each queue length is diminished by the removal of a single customer is equal to the amount that customer contributes to the queue length. In general, this assumption is quite accurate. In particular, it is asymptotically correct for very large N, and trivially correct for models with only a single customer (since it predicts that arrival instant queue lengths are zero). Thus, the approximation is guaranteed to be good at the two extremes. Experience with the technique has demonstrated that it also gives remarkably good results for intermediate populations. Since this error is well within the bounds of other discrepancies inherent in the computer system analysis

process (e.g., the accuracy of parameter values), the approximate MVA technique is satisfactory as a general solution technique.

Closed Model Example (Approximate Solution)

Table 6.3 lists the successive approximations for the device queue lengths obtained by applying this approximate solution technique to the same example used previously with the exact solution technique. The stopping criterion used was agreement in successive queue lengths within .001. The exact solution of the model is listed in the table for comparison. (Note once again the apparent anomaly caused by the fact that the class in this model is of type terminal. We initialize by distributing the customers equally among the three centers. As the iteration progresses, customers "disappear" from the table. At the conclusion of the iteration, the difference between the full customer population and the sum of the queue lengths at the centers represents the average number of users "thinking".)

iteration	Q_{CPU}	Q_{Disk1}	Q_{Disk2}	X	R
0	1.00	1.00	1.00		
1	.1390	.4826	.3102	.1379	6.7583
2	.0988	.4150	.2436	.1495	5.0659
3	.0972	.4043	.2366	.1508	4.8950
4	.0972	.4024	.2359	.1510	4.8732
5	.0973	.4021	.2359	.1510	4.8700
exact solution	.0976	.3947	.2350	.1515	4.8020

Table 6.3 - Approximate MVA Computation

6.5. Theoretical Foundations

Separable queueing network models are a subset of the general class of queueing network models obtained by imposing restrictions on the behavior of the service centers and customers. The name "separable" comes from the fact that each service center can be separated from the rest of the network, and its solution evaluated in isolation. The solution of the entire network then can be formed by combining these separate solutions. In an intuitive sense, a separable network has the property that each service center acts (largely) independently of the others.

There are five assumptions about the behavior of a model that, if satisfied, guarantee that the model is separable. These are:

- *service center flow balance* — Service center flow balance is the extension of the flow balance assumption (see Chapter 3) to each individual service center: the number of arrivals at each center is equal to the number of completions there.
- *one step behavior* — One step behavior asserts that no two jobs in the system "change state" (i.e., finish processing at some device or arrive to the system) at exactly the same time. Real systems almost certainly display one step behavior.

The remaining three assumptions are called *homogeneity assumptions.* This name is derived from the fact that in each case the assumption is that some quantity is the same (i.e., homogeneous) regardless of the current locations of some or all of the customers in the network.

- *routing homogeneity* — To this point we have characterized the behavior of customers in the model simply by their service demands. A more detailed characterization would include the routing patterns of the jobs, that is, the patterns of centers visited. Given this more detailed view, routing homogeneity is satisfied when the proportion of time that a job just completing service at center j proceeds directly to center k is independent of the current queue lengths at any of the centers, for all j and k. (A surprising aspect of separable models is that the routing patterns of jobs are irrelevant to the performance measures of the model. Thus, we will continue to ignore them.)
- *device homogeneity* — The rate of completions of jobs from a service center may vary with the number of jobs at that center, but otherwise may not depend on the number or placement of customers within the network.
- *homogeneous external arrivals* — The times at which arrivals from outside the network occur may not depend on the number or placement of customers within the network.

These assumptions are sufficient for the network to be separable, and thus to be evaluated efficiently. However, the specific solution algorithms we have presented thus far require one additional assumption, which is a stronger form of the device homogeneity assumption:

- *service time homogeneity* — The rate of completions of jobs from a service center, while it is busy, must be independent of the number of customers at that center, in addition to being independent of the number or placement of customers within the network.

The weaker of the two assumptions, device homogeneity, permits the rate of completions of jobs from a center to vary with the queue length there. Centers with this characteristic are called *load dependent* centers. A delay center is a simple example of a load dependent center, since the rate of completions increases in proportion to the number of customers at the

center. Service time homogeneity asserts that the rate of completions is independent of the queue length. Centers with this characteristic are called *load independent.* The queueing centers we have described so far are examples of load independent centers. The particular versions of the MVA algorithms presented in this chapter are applicable only to networks consisting entirely of load independent and delay centers. In Chapters 8 and 20 we discuss the modifications necessary to accommodate general load dependent centers.

Although the assumptions above are necessary to prove mathematically that the solution obtained using Algorithm 6.2 is the exact solution of the model, they need not be satisfied exactly in practice for separable models to provide good results. Experience has shown that the accuracy of queueing network models is extremely robust with respect to violations of these assumptions. Thus, while no real computer system actually satisfies the homogeneity assumptions, it is rare that violations of these assumptions are a major source of inaccuracy in a modelling study. More typically, the problems encountered in validating a model result from an insufficiently accurate characterization by the model at the system level, usually because of inaccurate parameter values for service demands or workload intensities. The only important exceptions to this are cases in which the limitations on the structure of the model imposed by the assumptions required for separability prohibit representation of aspects of the computer system important to performance (for example, the modelling of memory constraints or priority scheduling). In these cases, we would like models that are as easy to construct and to evaluate as separable networks, but that also represent the "non-separable" aspects of the computer system. In Part III of this book we show that collections of separable models evaluated together (typically iteratively) provide just such tools. Thus, separable models not only are adequate simple models of computer systems, but also are the basic building blocks out of which more detailed models can be constructed.

6.6. Summary

In this chapter we have examined the construction and evaluation of single class, separable queueing network models. Separable models have the following desirable characteristics:

- *efficiency of evaluation* — Performance projections can be obtained from separable models with very little computation. General networks of queues require so much computation to evaluate that they are not practical tools.

- *accuracy of results* — Separable models provide sufficiently accurate performance projections for the majority of modelling studies. We have described a number of case studies to illustrate this point. For the most part, the inaccuracy inherent in establishing parameter values and in projecting workload growth dominates the inaccuracy inherent in separable models. Thus, there is little motivation to look for more accurate models.
- *direct correspondence with computer systems* — The parameters of separable models (service centers, workload types, workload intensities, and service demands) correspond directly to a high level characterization of a computer system. Thus, it is easy to parameterize these models from measurement data in constructing a baseline model, and it is relatively simple to alter the parameters in an intuitive way to reflect projected changes to the computer system in the model.
- *generality* — In cases where the restrictions required in the construction of separable models exclude an important aspect of a computer system from being represented in an individual separable model, collections of separable models can be used. Thus, separable models are the basic tool that we will use throughout the book as we extend our models to include increasingly detailed aspects of computer systems.

We have studied single class separable models in this chapter because they form a natural bridge between the bounding models of Chapter 5 and the more detailed multiple class models of Chapter 7. Important characteristics of single class models in this regard are:

- *ability to project performance* — Single class models contain sufficient detail that performance estimates, rather than performance bounds, can be projected.
- *simplicity* — Single class models are the simplest models for which this is true: the simplest to define, parameterize, evaluate, and manipulate. In light of this, they are the models of choice in situations where they are sufficiently detailed to answer the performance questions of interest.
- *pedagogic value* — The more detailed multiple class models presented in Chapter 7 are considerably more cumbersome notationally than single class models, but actually are very simple extensions of these models. Thus, an understanding of single class models aids in understanding the definition, parameterization, and use of multiple class models.

In the next chapter we extend our modelling capabilities to accommodate systems containing several distinct workload components, which we represent using multiple class, separable queueing network models.

6.7. References

Single class models originally were viewed in the stochastic setting. Jackson [1963] described networks of exponential queues and showed that their solution was separable. Gordon and Newell [1967] obtained similar results for closed networks, and showed that the state probabilities have a simple solution known as "product form".

Buzen [1973] introduced the first efficient evaluation algorithm for closed models. Reiser and Lavenberg [1980] developed the exact mean value analysis algorithm described here. The fact that $A_k(N) = Q_k(N-1)$ in separable queueing networks was established by Sevcik and Mitrani [1981], and independently by Lavenberg and Reiser [1980]. The approximate MVA algorithm is based on work by Bard [1979] and Schweitzer [1979]. Chandy and Neuse [1982] and others subsequently have developed related approximations.

The case studies of Sections 6.3.1, 6.3.2, and 6.3.3 were carried out by Scherr [1967], Lipsky and Church [1977], and Levy [1979], respectively. Scherr's monograph is the source of Figure 6.2, and Levy's paper the source of Figure 6.5.

Denning and Buzen [1978] discuss the homogeneity assumptions in greater detail than we have presented.

[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.

[Buzen 1973]
Jeffrey P. Buzen. Computational Algorithms for Closed Queueing Networks with Exponential Servers. *CACM 16,9* (September 1973), 527-531.

[Chandy & Neuse 1982]
K. Mani Chandy and Doug Neuse. Linearizer: A Heuristic Algorithm for Queueing Network Models of Computing Systems. *CACM 25,2* (February 1982), 126-133.

[Denning & Buzen 1978]
Peter J. Denning and Jeffrey P. Buzen. The Operational Analysis of Queueing Network Models. *Computing Surveys 10,3* (September 1978), 225-261.

[Gordon & Newell 1967]
W.J. Gordon and G.F. Newell. Closed Queueing Networks with Exponential Servers. *Operations Research 15* (1967), 244-265.

[Jackson 1963]
J.R. Jackson. Jobshop-like Queueing Systems. *Management Science 10* (1963), 131-142.

[Lavenberg & Reiser 1980]
S.S. Lavenberg and M. Reiser. Stationary State Probabilities of Arrival Instants for Closed Queueing Networks with Multiple Types of Customers. *Journal of Applied Probability* (December 1980).

[Levy 1979]
Allan I. Levy. Capacity Planning with Queueing Network Models: An IMS Case Study. *Proc. CMG X International Conference* (1979), 227-232.

[Lipsky & Church 1977]
L. Lipsky and J.D. Church. Applications of a Queueing Network Model for a Computer System. *Computing Surveys 9*,3 (September 1977), 205-222. Copyright © 1977 by the Association for Computing Machinery.

[Reiser & Lavenberg 1980]
M. Reiser and S.S. Lavenberg. Mean Value Analysis of Closed Multichain Queueing Networks. *JACM 27*,2 (April 1980), 313-322.

[Scherr 1967]
Allan L. Scherr. *An Analysis of Time-Shared Computer Systems.* Research Monograph No. 36, MIT Press, 1967. Copyright © 1967 by the Massachusetts Institute of Technology.

[Schweitzer 1979]
P. Schweitzer. Approximate Analysis of Multiclass Closed Networks of Queues. *Proc. International Conference on Stochastic Control and Optimization* (1979).

[Sevcik & Mitrani 1981]
K.C. Sevcik and I. Mitrani. The Distribution of Queueing Network States at Input and Output Instants. *JACM 28*,2 (April 1981), 358-371.

6.8. Exercises

1. Suppose we wish to plot response time estimates obtained from a separable single class queueing network model for all populations from 50 to 75 online users:
   a. If the exact solution technique were used, how many applications of the algorithm would be required to compute performance measures for all 26 populations?

b. Using the approximate solution technique, how many applications of the algorithm would be required?

Suppose that users of this system overlapped the preparation of each request with the processing of the previous request, so that effective think time varied with system response time, and thus with the user population. (For instance, average think time might be 10 seconds with 50 active users, and 8 seconds with 65 active users.)

c. Under this assumption how many applications of each algorithm would be required?

d. Why would it be incorrect simply to modify Algorithm 6.2 (the exact solution technique) so that the think time, Z, was a function of the user population?

2. Exercise 4 in Chapter 5 asked you to graph asymptotic and balanced system bounds for a simple model in two cases: batch and terminal workloads. Use Algorithm 6.2 to compute throughput and response time for these cases for values of N from 1 to 5. Use Algorithm 6.3 for $N=5$ and $N=10$. Compare these results with the bounds obtained previously.

   a. How much additional effort was required to parameterize the single class model in comparison with the bounding models?

   b. How do the techniques compare in terms of computational effort?

   c. How do the results of the techniques differ in terms of their usefulness for projecting performance? In terms of your confidence in the information that they provide?

3. Implement Algorithm 6.3, the approximate mean value analysis solution technique. Repeat Exercise 2 twice: once using this implementation, and once using the Fortran implementation of Algorithm 6.2 (exact mean value analysis) contained in Chapter 18. Compare the results.

4. Modify the program given in Chapter 18 to allow delay centers, and to allow classes of transaction type.

5. Use the modified program, as follows:

   a. Evaluate a model with three centers with service demands of 8, 5, and 4 seconds, and a transaction class with arrival rate .1 requests/second.

   b. Using the response time obtained in (a), calculate an appropriate think time for use in an equivalent model with the transaction class replaced by a terminal class with 10 users.

c. Evaluate the model constructed in (b).

d. Explain the differences between the performance measures obtained in (a) and (c).

6. Use the arrival instant theorem to show that in a balanced model (one in which the service demands at all centers are equal to $D_k = D/K$), system throughput is given by:

$$X = \frac{N}{N+K-1} \times \frac{1}{D_k}$$

(This result is the basis of balanced system bounds, as presented in Chapter 5.)

7. Both the exact and the approximate MVA algorithms involve four key equations (6.1 through 6.4).

a. For each of these four equations, provide an intuitive justification in a few words.

b. In a few sentences, describe how the exact MVA algorithm is constructed from these four components.

c. In a few sentences, describe how the approximate MVA algorithm is obtained from the exact algorithm.