Chapter 12

Existing Systems

12.1. Introduction

In this chapter we discuss the construction of baseline models of existing systems. This activity relies on knowledge of the hardware, software, workload, and monitoring tools associated with the system under study. It also requires access to information recorded by accounting and software monitors during system operation. Here, we describe general approaches applicable to a variety of systems. In Chapter 17, we illustrate these approaches with an example based on a specific system (IBM's MVS) and a specific monitoring tool (RMF).

In Chapter 4 we divided the inputs of queueing network models into three groups: the *customer description*, the *center description*, and the *service demands*. The structure of the present chapter reflects this division.

Section 12.3 is devoted to the customer description: the correspondence of the workload components of the system to the customer classes of the model. In specifying the values of the customer description parameters, we are answering questions such as:

- How many customer classes are required?
- Of what type (transaction, batch, or terminal) should each class be?
- What should be the workload intensity value (λ, N, or N and Z) for each class?

Section 12.4 is devoted to the center description: the correspondence of the resources of the system to the service centers of the model. In specifying the values of the center description parameters, we are answering questions such as:

- What devices and subsystems should be included in the model?
- How should each of these entities be represented (e.g., as a queueing center, a delay center, or an FESC)?

Section 12.5 is devoted to the service demands: the description of the interactions between customers and centers. In specifying the values of the service demand parameters, we are answering the question:

– What proportion of the measured usage of each device should be attributed to the customers of each class?

We precede these three sections, in Section 12.2, with a survey of the information used to parameterize queueing network models: its types, its sources, and how it can be managed. We follow these sections, in Section 12.6, with a discussion of the validation of baseline models, indicating reasonable tolerances for various performance measures.

There is little reason to construct a model of an existing system unless this model is to be used for performance projection. Consequently, we cannot completely separate the task of constructing a baseline model of an existing system (the subject of this chapter) from the task of using the model to project performance for an evolving system (the subject of Chapter 13). Our (somewhat artificial) separation between the two tasks will be the following: problems that arise from limitations or shortcomings of current monitoring tools and techniques will be treated in this chapter, while problems that would persist even with ideal monitoring capabilities will be deferred to the next chapter.

12.2. Types and Sources of Information

The information required to specify parameter values for a queueing network model of an existing system includes *static* information about the system configuration and *dynamic* information extracted from records produced during system operation by various monitoring packages. Some information is recorded for purposes of accounting, while other information is recorded explicitly for performance evaluation purposes. Software packages of varying degrees of sophistication are available for storing, analyzing, and reporting the information recorded during system operation. In this section, we discuss briefly the information needed, how it can be obtained, and how it can be managed. Our intention is not to be comprehensive, but rather to highlight points of particular relevance to the construction and use of queueing network models.

One type of information required is a description of the hardware and software of the system. With respect to hardware, this information includes an enumeration of the components of the system (processors, channels, storage devices, communication devices, etc.) and an indication their interconnections (e.g., the paths over which data can be moved from a particular storage device to memory). With respect to software, this information includes the operating system in use, and the values of parameters that influence resource allocation. Examples of such parameters include CPU scheduling priorities for various workload components, placement of files on storage devices, etc.

This system description is relatively static, in that it changes only week to week or month to month. The information it provides about the hardware suggests what resources should be represented as centers in the model. The information it provides about the software and operating policies suggests appropriate modelling assumptions and helps in the interpretation of measurement data.

Another type of information that is required is recorded dynamically during system operation by various monitors. Accounting monitors write records at the termination of batch jobs or interactive sessions, indicating the system resources consumed by the job or session (CPU seconds, I/O operations, memory residence time, connect time, etc.). Software performance monitors write records describing resource usage and performance status from another point of view. At specified intervals, queue lengths or device status indicators may be *sampled* and the results written in a record. Also, certain *events* that are considered significant (such as swapping a customer out of main memory) may be documented in a record.

Because of their volume and their encoding, the records produced by accounting and software monitors are not usable directly. Rather, they must be processed by reporting routines that produce summary information for a specific purpose (e.g., accounting, workload forecasting, performance modelling). Most accounting and software monitors are packages that include both a *recording* component and a *reporting* component. For example, accounting records are written for each unit of work processed, and an accounting program periodically passes over the recent accounting records to determine charges for each account. Similarly, software monitors write records at certain events or sampling intervals, and a post-processor later examines the records and produces reports organized to aid system tuning and performance evaluation.

The reports produced by accounting and software monitors usually are organized in one of two ways. Some reports are *class based*: they organize information by user or by workload component. Other reports are *resource based*: they organize information by system resource. Monitors that reliably break down resource usage by both workload component and resource are not used commonly in most systems. (Those that exist cause prohibitively high monitoring overhead.) Much of the effort in parameterization, as described in Sections 12.3 to 12.5, arises from the need to surmount the inadequacies of commonly available measurement information. As software monitors are improved, the parameterization task will become less burdensome, and some of the techniques described in this chapter will become unnecessary.

When using a reporting routine to obtain information, it is necessary to specify the interval of time over which information is to be gathered. Generally, it is appropriate to run the monitor during peak loads, as these present the most significant performance problems. The duration of the observation interval should be long enough that *end effects* do not significantly affect the accuracy of the measurements. End effects are measurement errors caused by the fact that some customers are processed partly within and partly outside of the observation interval. In particular, it is typical to assume that the system operates in flow balance over the measurement interval, so that the job arrival and completion rates are equal. However, because some jobs arrive but do not complete during the interval, and other jobs arrive before but complete during the interval, flow balance may not hold. Clearly, measurements obtained from longer observation intervals are affected less by these end effects than are shorter intervals. Typically, observation intervals of thirty to ninety minutes are appropriate for obtaining software monitor data. If monitoring overhead is a concern, shorter intervals can be used, but the danger of anomalies is increased.

Other sources of useful information include hardware monitors and monitors specialized for particular application subsystems (such as database or telecommunications subsystems). Hardware monitors, because they are "external observers" of the system, obtain accurate measurements and do not perturb system operation. They are incapable, however, of associating resource usage with workload components. The specialized application subsystem monitors are helpful in assessing the performance of subsystems whose autonomy from the host operating system prevents standard monitors from being able to record their activity. (For example, special monitors are needed for IBM's IMS database system because RMF does not record information about individual IMS transactions.) While any information that is available from hardware and specialized application subsystem monitors should be exploited, our discussion in this chapter will be restricted to the kinds of information that are commonly reported in most medium or large computer installations.

Table 12.1 summarizes the information typically available from various sources. Information from different sources (accounting and software monitors, or even two different software monitors) may be based on different underlying assumptions. For this reason, and also because of end effect anomalies, information from different sources may appear to be contradictory. For example, consider a small interactive system in which monitors report that in a thirty minute observation interval:

- 7200 transactions were processed
- average response time was three seconds
- the sum of the queue lengths at the CPU and all disks was 18

We would conclude that throughput during the observation interval was:

$$\frac{7200 \text{ transactions}}{1800 \text{ seconds}} = 4 \text{ transactions/second}$$

type	**information provided**
system description	hardware configuration operating system (and version) resource allocation and scheduling strategies tuning parameter values
accounting monitor	CPU usage, by workload component logical I/O operation count, by workload component customer completions, by workload component
software monitor	measured busy time, by device physical I/O operation count, by device average queue length, by device throughput, by workload component average response time, by workload component
hardware monitor	observed busy time, by device

Table 12.1 – Sources of Information

Because the observation interval is long relative to the average response time, we could be confident that end-effects would not lead to significant errors in the estimates of throughput or response time. Considering Little's law, however, we would find the sum of the queue lengths (18) to be much higher than expected from the product of throughput (4 transactions/second) and response time (3 seconds). One possible explanation for such a situation is that the queue lengths include system tasks that are not counted in either the throughput or response time calculations. On the other hand, if the sum of the queue lengths had been reported as 8 (and other values remained the same), then Little's law would reveal a discrepancy in the other direction. A possible explanation for the second case would be that requests were queueing for admission to memory, thus spending a significant part of their response time where they were not included in the queue length of any device. The fundamental laws presented in Chapter 3 can be used to detect such apparent contradictions. System intuition and careful thought is required to resolve them.

Enhanced awareness of the problems of configuration management and capacity planning has led recently to some encouraging progress in the use and management of system measurement data. First, special reporting routines tailored to the requirements of queueing network modelling have been developed for some systems. These routines analyze records produced by existing accounting and software monitors. Some are capable of defining a queueing network in a format directly acceptable by particular queueing network modelling software packages.

While these routines are a great aid, intervention by an analyst still is necessary in most cases to obtain a validated model. This is true because of inadequacies in the measurement data, and the fact that the analyst's knowledge of the system is not available to the automated routine. (Further discussion of such routines appears in Chapter 16.)

Second, some of the newer reporting routines have been generalized to be capable of using and contributing to a *performance database*. The records written by various monitors constitute a rudimentary performance database. Merely organizing the records according to their types and source makes them easier to use. The utility of the database is further enhanced, however, if it is extended to include aggregated information produced by reporting routines. There are several advantages to maintaining such a performance database. For one, long-term trends can be examined if information aggregated on a month by month basis is included in the database. Also, information intended for management planning can be isolated from the more technically oriented information intended for system tuning. Finally, by having various aggregations of monitoring information available in a database, the need for regular printed reports is substantially reduced.

12.3. Customer Description

Most large computer systems have workloads consisting of several identifiable components. Performance studies often are intended to assess performance of each workload component, since system-wide average values for throughput and response time have little significance in systems that include such diverse workload components as background batch and foreground transaction processing. There are several goals to meet in deciding how to assign the workload components of the system to the customer classes of a queueing network model:

- Classes should consist of customers whose service demands are of comparable magnitude and similar balance across service centers, since input parameters to the model for all customers in the same class are identical. (For example, I/O bound customers should not ordinarily be in the same class as CPU bound customers.)
- Classes must distinguish workload components for which independent performance projections are desired as outputs of the model. (For example, if response time to database queries is of concern, then database queries should not be grouped in a single class with other workload components.)

- Classes may be made to correspond to accounting and performance groups. This facilitates the calculation of various parameter values, since accounting data is organized by accounting group.
- Classes may be used to distinguish work generated by various organizational units (e.g., divisions of a company). This permits unit-specific performance projections, and facilitates later modification analysis (since workload forecasts frequently are made on an organizational unit basis).

A first step in identifying customer classes is to group portions of the workload according to whether they are best represented as batch, terminal, or transaction types. Often, the nature of a workload component suggests an appropriate type: if requests arrive at a constant rate, then transaction; if requests are generated by a set of users that await the completion of service to one request before generating another, then terminal; if the number of active requests is constant, then batch. Variations are possible, though, especially in conducting a modification analysis. As one example, a workload component might in fact consist of users at terminals, but for planning purposes its intensity might be described in terms of a request arrival rate. In this case, the use of a transaction type might be appropriate. As another example, a system might have many workload components, only a few of which are of interest. The presence of the other components might be reflected in the model by a single "aggregate" class of transaction type (so that its throughput is guaranteed to equal the measured value).

Within each type of customer class, further separation of workload components may be desirable. Batch work of different priorities may be represented as distinct classes. Different interactive systems (e.g., APL and TSO in an IBM environment) may be treated as separate terminal classes. If trivial transactions (such as simple editing commands) can be distinguished from substantive transactions (such as complex database queries), then different classes can be used to distinguish the two groups.

The queueing network model input parameter C is simply the number of customer classes, determined according to the guidelines suggested above. Models of simple systems typically have just one or two classes, while models of complex multi-purpose systems may have eight or more. In some special situations it is useful to have a very large number of classes — say, twenty to forty.

One example of a situation in which a large number of classes was used is a model developed for projecting the performance of a hospital information system used in many hospitals. There were roughly thirty major transaction types (admit-patient, order-blood-test, set-dietary-restriction, etc.) each one of which was represented as a separate customer class. In this way, the arrival rate of each transaction type and the priority assigned to the transaction type (reflecting its urgency in a particular hospital) could be represented directly in the model. The hospitals using the system differed substantially in size and in the hardware on which they ran the system. Also, they differed significantly in the particular mix of transactions that were processed. The model proved useful in configuration design. The response times for various transaction classes could be related to the arrival rates and priorities of the classes for various contemplated hardware configurations.

Having identified each workload component to be represented as a distinct customer class and determined the type of that class, the next step is to establish the workload intensity of each class. For a transaction class, the workload intensity is the transaction arrival rate. Over a reasonably long observation interval in a system that is not saturated, the arrival rate is essentially the same as the completion rate. Consequently, an estimate for the arrival rate of class c is:

$$\lambda_c = \frac{\textit{measured completions of class } c}{\textit{length of measurement interval}}$$

For a batch class, the workload intensity is given by the average number of batch customers active. An estimate for N_c, the number of class c customers, can be obtained in several ways:

- If jobs are processed in a fixed number of regions and memory queueing times are high (so that it is known that each region is busy throughout most of the observation interval), then N_c is the number of processing regions.
- If the software monitor provides an estimate of the average multiprogramming level of the class over the observation interval by sampling, then N_c can be taken to be that estimate.
- If accounting data provides the residence time of each job in the central subsystem, then N_c can be estimated by:

$$N_c = \frac{\sum\limits_{\substack{\textit{class } c \\ \textit{jobs}}} \textit{measured job residence time}}{\textit{length of measurement interval}}$$

  (This alternative is impractical without the use of a reduction package capable of automatically extracting this information from accounting records.)

For a terminal class, workload intensity is specified by the number of active terminals, N_c, along with the average think time, Z_c. Three possibilities for estimating N_c for terminal classes correspond directly to the three methods used for batch classes:

- If terminals connect to the system through a limited number of ports, and if all ports are busy throughout most of the observation interval, then N_c is the number of ports.
- If the software monitor provides the average number of active terminals over the observation interval, then N_c can be taken to be that number.
- If accounting data includes session lengths, then N_c can be estimated (over an observation interval that is long relative to average session length in order to restrict end effects) by:

$$N_c = \frac{\sum\limits_{\substack{class\ c \\ sessions}} measured\ session\ length}{length\ of\ measurement\ interval}$$

The average think time of a terminal class often is one of the most difficult input parameters to estimate. There are several reasons. First, there are differing views of when think time starts and ends. We will adopt the one in which it starts with the arrival of the first character of a response from the system, and ends when the last character of the next request to the system is entered. Second, some systems allow a stream of commands to be entered without awaiting responses. Such systems can cause think times (as defined above) to be negative! Third, some think times become so long that they actually represent a loss of an active terminal. (This occurs when terminal users interrupt their work without logging off.) Fourth, average think time seldom is measured directly by performance monitors. Consequently, the best estimate of think time often is obtained by estimating Z_c from the response time law:

$$Z_c = \frac{N_c}{X_c} - R_c$$

where N_c is estimated as described above, and X_c and R_c are measured values. Because there often is less confidence in the estimate of think time than in the estimates of other parameters, it may be desirable to test the sensitivity of the model to this value.

When memory constraints are imposed on transaction or terminal classes, it is necessary to specify the capacity associated with each *domain* so that the modelling approach of Section 9.3 can be used. The capacity of each domain typically is known from the system description. Whether or not the domain was filled to capacity in a particular measurement interval is revealed by comparing the average number active among classes assigned to the domain (as reported by a monitor) to the domain capacity.

12.4. Center Description

The service centers of a queueing network model correspond to significant points of congestion or delay in the system. There are many ways of representing system resources by a set of service centers. Here we suggest only the most widely accepted methods, which have proven successful in a large number of modelling studies.

For systems with single CPUs and for tightly-coupled multiprocessors, a single service center is used to represent the CPU(s) in the queueing network model. Loosely-coupled multiprocessors are modelled by including one service center per processor. Front end communications processors and back end database machines also may be represented as separate service centers.

The representation of disk subsystems can be done in a variety of ways. (See the discussion in Chapter 10.) A number of components are involved in each disk I/O operation. The modelling approach that has proven most successful, however, is to use a single service center to represent each disk. Congestion due to other I/O subsystem components is represented by calculating an appropriate *effective service demand* for each center.

Other peripheral devices can be represented more simply than disks. Because tape drives are not capable of operation independent of the channel, a group of tape drives on a channel can be represented by a single service center. The service demands at the center can be established using channel utilization only, and ignoring the individual tape drives.

Unit record equipment typically is ignored in constructing queueing network models. This is justified in many systems because *spooling* makes the use of unit record devices asynchronous. Similarly, terminal controllers typically are not represented. If delays in the communications front end are thought to be important in a particular study, then a special approach must be used. This might involve a hierarchical model in which a conventional central subsystem model is evaluated, and then the delays due to communication are represented in a high-level model that includes an FESC representing the central subsystem.

12.5. Service Demands

The final set of values needed to parameterize a queueing network model are the service demands at each center of the customers belonging to each class. Obtaining these values can be a difficult and time consuming process. As a practical consideration, it is important to concentrate on obtaining accurate estimates for the most heavily utilized centers, because a small error in estimating the service demands at the bottleneck center will affect performance projections more than a much larger error at a lightly utilized center.

In estimating service demands, the three center types (delay, FESC, and queueing) are treated differently.

Delay centers have service demands that represent a delay that is not caused by congestion (e.g., a propagation delay in a communication network). It usually is not difficult to determine appropriate values for delay centers. In addition, errors in the service demands at delay centers are not "magnified" by queueing delay calculations when the model is evaluated.

For FESCs, the load dependent service rates can be determined in many ways, as described in Chapter 8. Two major approaches are evaluating low-level queueing network models (as illustrated in Chapter 9 for the case of memory constraints) and considering hardware characteristics (as illustrated in Chapter 11 for the case of tightly-coupled multiprocessors).

The remainder of this section is devoted to the case of queueing centers, by far the most common center type in queueing network models. Conceptually, estimating service demands for queueing centers is straightforward: at the conclusion of the measurement interval, the measured busy time for each class at each device is divided by the number of system completions for the class. In practice, however, two difficulties arise:

- In the multiple class case, the available data frequently is insufficient to apportion the measured busy time among the classes with certainty. The reasons and the remedies differ for various devices and various systems.
- A portion of the busy time attributed to each class is *intrinsic* to that class: its basic processing and I/O requirements. The remainder consists partly of *service demand inflation* and partly of *overhead.* Service demand inflation, introduced in Chapter 10, is the component of measured disk busy times due to contention in the I/O subsystem. (There is no service demand inflation for processors.) Overhead is work done by the operating system "on behalf of" the customers of the class. Part of the overhead component is *fixed*, in that it does not depend on system congestion (e.g., the CPU service required to initiate user I/O operations), and part of it is *variable* and typically increases with system load (e.g., paging I/O). In a baseline model these distinctions do not matter, but in conducting a modification analysis they can be crucial, for the service demand inflation and variable overhead components of the model usually change in a new environment.

This section is devoted to the first of these two difficulties: apportioning measured busy time among the various classes. We defer our discussion of the second difficulty to Chapter 13. The reader should understand, however, that while the techniques used to adjust the service demand inflation and variable overhead components of service demands are not required until projecting performance for an evolving system, they should be validated by examining several measurement intervals using the baseline model of the existing system.

Our discussion is organized into two subsections, the first devoted to processors and the second to I/O.

12.5.1. Estimating Processor Service Demands

Since the CPU typically is a heavily utilized resource, it is important to determine accurately the service demands of the various classes there. As noted in Table 12.1, monitor data often includes the CPU usage and the number of customer completions for each workload component. Unfortunately, the quotient of these quantities turns out in practice to yield a poor estimate of CPU service demand. The reason is that the CPU usage reported on a per class basis often fails to capture significant amounts of CPU activity. More specifically, the sum of the CPU busy times reported on a per class basis is likely to be considerably less than the total CPU busy time reported by a monitor that does not attempt to distinguish among classes. The ratio of attributed CPU usage for a class to the total CPU busy time due to activities initiated by that class is known as the *capture ratio.* Capture ratios typically range from .85 down to .40 for various systems and various workload components. For a particular system, the overall capture ratio can be estimated as suggested above: by dividing the sum of the CPU busy times reported on a per class basis (often by an accounting monitor) by the total CPU busy time reported by a monitor that does not attempt to distinguish among classes (often by a software monitor).

In the case of single class models, dividing the estimate of total CPU busy time from software monitor data by the estimate of total customer completions from either accounting or software monitor data will yield a good estimate for CPU service demand. In the case of multiple class models, though, techniques must be devised to apportion the unattributed CPU busy time among classes. This process has three steps:

- calculate the unattributed busy time during the interval
- decide how much to attribute to each class
- compute how much to attribute to each customer of each class

The second of these steps is the interesting one, and will be addressed in the paragraphs that follow.

Consider a system with a workload consisting of two components: batch jobs and interactive users. Assume that information comparable to that listed in Table 12.1 has been obtained. Let f_{BATCH} and f_{INTER} be (unknown) factors by which the attributed CPU busy time for each class must be multiplied so that all measured CPU busy time is attributed to some class. (Observe that f_c is the inverse of the capture ratio for class c.) This leads to the equation:

$$B_{CPU} = f_{BATCH} \times A_{BATCH,CPU} + f_{INTER} \times A_{INTER,CPU}$$

where $A_{c,CPU}$ is the CPU usage attributed to class c, and B_{CPU} is the total measured CPU busy time.

To determine unique values for f_{BATCH} and f_{INTER} we must establish a relationship between them in addition to this equation. Several possibilities exist:

- Assume that the ratio of total CPU time to attributed CPU time is the same for each class, yielding:

$$f_{BATCH} = f_{INTER} = \frac{B_{CPU}}{\left[A_{INTER,CPU} + A_{BATCH,CPU}\right]}$$

- Since the unattributed CPU busy time is likely to be overhead, use class based information on activities likely to cause CPU overhead (such as paging rate, swapping rate, spooling, user I/O, and job initiations) to determine a relative measure of total overhead for each class. For instance, assuming that overhead is due almost entirely to page fault handling, and letting OV_c (the relative overhead of class c) be the measured number of pages transferred because of class c faults, we have:

$$f_{INTER} = 1 + \frac{\frac{OV_{INTER}}{OV_{INTER} + OV_{BATCH}} \times \left[B_{CPU} - \left[A_{INTER,CPU} + A_{BATCH,CPU}\right]\right]}{A_{INTER,CPU}}$$

The second approach is the more reasonable. Unfortunately, more than one factor inevitably contributes to overhead. Thus, OV_c is better defined as the weighted sum of several factors:

$$OV_c = \sum_{all\ factors\ i} weight\ i \times factor\ i_{\ c}$$

When one attempts to apply this approach in practice, two common problems are apt to be encountered:

- Even for a single measurement interval, it may be difficult to determine which factors to consider, and what weights to assign to these factors. Iteration inevitably is required: estimate weights, calculate service demands, evaluate model, re-estimate weights, etc.
- If one truly is to have confidence in the weights selected, then data from a number of measurement intervals must be considered, and weights must be found that yield good model results when applied to each set of data. An ad hoc approach can be adopted, or linear regression techniques can be used.

Once f_{BATCH} and f_{INTER} have been determined, the service demands of the two classes can be estimated by the equation:

$$D_{c,CPU} = \frac{f_c \times A_{c,CPU}}{measured\ class\ c\ completions}$$

Note that the service demands determined in this way include intrinsic service, fixed overhead, and an amount of variable overhead that reflects the degree of system congestion in the interval covered by the measurement data.

12.5.2. Estimating I/O Service Demands

I/O activity in most current computer systems is dominated by operations on direct access storage devices (fixed head, movable head, and electronic disks). Tape I/O and I/O for staging data to and from mass storage devices plays a secondary role. Other types of peripheral devices typically are inconsequential with respect to performance. Our discussion in this section focuses on disk I/O, reflecting its importance.

In Section 10.7 we described how the lengths of certain portions of disk service requirements (seek, latency, rotation, and transfer) could be established from system knowledge (e.g., device characteristics) and measurement data. We assumed that both the visit counts and the service times per visit for each class at each disk were known. In this section, we suggest a method for determining these quantities. First we consider the visit counts, then the service times.

We distinguish two ways of viewing I/O operations. *Physical I/O* operations correspond to activations of I/O subsystem components to transfer data to or from peripherals. *Logical I/O* operations correspond to operating system calls by customers requesting access to blocks of information. For a number of reasons physical and logical I/O operations do not correspond directly to one another. Sometimes, a logical I/O operation may not result in a physical I/O operation; for example, a logical I/O operation may request access to a block of information that already is in memory. Sometimes, a logical I/O operation may result in several physical I/O operations; for example, errors detected in reading or writing a block may cause operations to be retried.

It is the physical I/O operations that correspond to the visit counts, but physical operations seldom are reported on a per class basis. Typically, logical I/O operations are broken down by class but not by device (often by an accounting monitor), while physical I/O operations are broken down by device but not by class (often by a software monitor).

The first step in confronting this situation is to estimate the ratio of physical to logical I/Os for each class. We now restrict consideration to a set of disk drives. Let P_k denote the physical I/Os at disk k, and let L_c denote the logical I/Os of class c over the set of disks. (Some monitors fail to distinguish logical disk I/Os from other logical I/Os. In such cases, we are forced to make some assumption such as that the fraction of all logical I/Os that are directed to the disks is the same as the fraction of all physical I/Os that are directed toward the disks, which is presumed to be available from measurements.) We define g_c to be the ratio of physical to logical I/Os for class c. (The assumption that the ratio depends on class but not device is realistic in most systems.) Estimating the g_c is a problem analogous to estimating the f_c in the case of the CPU. Possible approaches include:

- Assume that g_c is the same for each class, so that:

$$g_c = \frac{\sum\limits_{all\ disks\ k} P_k}{\sum\limits_{all\ classes\ j} L_j}$$

- Use generally accepted ratios for standard types of workloads for the architectural family of the system.
- For a number of observation intervals, determine the values for the g_c that best satisfy the set of equations:

$$\sum_{all\ disks\ k} P_k(i) = \sum_{all\ classes\ c} g_c \times L_c(i)$$

  where (i) denotes values obtained during the i-th observation interval.

Once these g_c have been estimated, we proceed to determine the visit counts. In essence, we must satisfy the equations:

$$P_k = \sum_{all\ classes\ c} (measured\ class\ c\ completions) \times V_{c,k}$$

$$L_c = (measured\ class\ c\ completions) \times \sum_{all\ disks\ k} \frac{V_{c,k}}{g_c}$$

class	**device**		
	disk 1	disk 2	disk 3
BATCH	?	?	?
INTER	?	?	?

adjusted logical I/Os
$L_{BATCH} \times g_{BATCH}$
$L_{INTER} \times g_{INTER}$

physical I/Os	P_1	P_2	P_3

Table 12.2 – Physical Disk I/Os by Class and Device

Table 12.2 suggests a way of thinking about the problem of determining the number of physical I/Os by each class at each device, again for the case of two classes, batch (*BATCH*) and interactive (*INTER*). The central rows correspond to classes, while the central columns correspond to disks. The entry to be filled in at column k of row c is the number of physical I/Os by class c at device k ($V_{c,k}$ × *measured class c completions*). The information available, however, is only that the columns must add to P_k while the rows must add to $L_c \times g_c$. This provides a number of equations equal to the sum of the number of classes and the number of disks, whereas the number of $V_{c,k}$ values that we must estimate is equal to the product of these quantities. (For instance, in Table 12.2 there are five constraints corresponding to the two row sums and three column sums, but there are six $V_{c,k}$ values to be determined.) Consequently, we must use additional information to specify the $V_{c,k}$ values uniquely. Alternatives include:

- The simplest assumption, which can be used in the absence of any other information, is that all classes use the various disks in the same proportions:

$$\frac{V_{c,k}}{V_{c,k'}} = \frac{V_{c',k}}{V_{c',k'}} \quad \text{for classes } c \text{ and } c', \text{ and disks } k \text{ and } k'$$

- The software configuration portion of the system description frequently indicates the location of various key data sets: paging files, swapping files, catalogs, files devoted to various applications, etc. If a particular class is known not to use a device, then its visit count there can be set to zero. If a particular class is known to be the exclusive user of a device, then its visit count there can be set to the measured physical I/O count of the device divided by the measured number of completions of the class. The remaining visit counts can be resolved in a series of stages. At each stage, the distribution of I/Os for the class for which the *least* flexibility remains is determined.

- In some systems there are software monitors capable of observing directly the number of physical I/Os broken down by both class and device. Although such monitors cause too much overhead to be used continuously, they can be used over short intervals (e.g., 10 minutes) to obtain an indication of the distribution of physical I/Os by class and device.
- Occasionally, the breakdown of logical I/Os by device as well as by class is known. This additional information makes it possible to proceed with greater confidence. In particular, if we can assume that the ratio of physical I/Os to logical I/Os is the same for each class, then the physical I/Os at a particular device can be attributed to classes in the same proportions as are the logical I/Os.

We turn now to the problem of determining the $S_{c,k}$. It is customary to assume that, at any particular disk, all classes have the same service time per visit. With this simplification, the service times are given by:

$$S_{c,k} = S_k = \frac{B_k}{P_k}$$

Situations in which one class has a substantially larger service time at a disk than another class typically arise when the former class uses a much larger block size. In such cases, disk characteristics (transfer rates, rotation times, and seek time functions) can be used to estimate the ratios $S_{c,k}/S_{c',k}$, for each pair of classes c and c' that use the disk. Those ratios, together with the equation:

$$B_k = \sum_{\text{all classes } c} V_{c,k} \times S_{c,k} \times (\textit{measured class c completions})$$

allow unique determination of the $S_{c,k}$. In both the cases of equal and unequal service times across classes, the service demands are given by:

$$D_{c,k} = V_{c,k}\, S_{c,k}$$

We now consider briefly the estimation of service demands for tape devices. As noted in an earlier section, it generally is appropriate to represent the tape channels rather than the individual drives. Further, it generally is appropriate to model all classes as using the various tape channels in the same proportions (although different classes will have different total amounts of tape I/O activity). Thus, the visit counts are given by:

$$V_{c,k} = \frac{L_c}{\sum_{\text{all classes } j} L_j} \times P_k \times \frac{1}{\textit{measured class c completions}}$$

where the P_k and L_c now are measured physical tape I/Os at center k and logical tape I/Os of class c, respectively. Assuming that all classes use essentially the same block size (so that they have the same service times), the service demands are given by:

$$D_{c,k} = V_{c,k} \frac{B_k}{P_k}$$

If block sizes differ significantly among classes, then service demands can be determined in a manner analogous to that suggested above for disks with class-dependent service times.

12.6. Validating the Model

Once values are established for all inputs, the model can be evaluated using the algorithms described in Part II, extended as described in Part III. This evaluation yields, for each class, estimates of system throughput and response time, and of device residence time, utilization, and queue length.

Model validation involves comparing these estimates with the measured values of the corresponding quantities. A model can be considered "validated" when it has been demonstrated that, in several (or many) measurement intervals, the differences between the estimates produced by the model and the measured quantities are sufficiently small.

In choosing observation intervals for use in validating the model, it is desirable to look ahead to the types of system changes to be investigated with the model. If the model is to be used to investigate the effect of an increased workload intensity, then the model should be validated on observation intervals representing a range of workload intensities. Similarly, if an increase in the size of main memory is to be considered, it is beneficial to validate the model on several different memory sizes. This could be done in a number of ways. Scheduling parameters could be adjusted to keep the number of active customers artificially low (thus underutilizing the memory). Alternatively, a portion of the memory could be disabled during an observation interval.

The correspondence between model estimates and measured quantities depends on several factors. Single class models can be validated with higher precision than multiple class models because their input parameter values can be determined from measurement data with greater accuracy. Some performance measures can be matched more easily than others. In validating multiple class models, it seldom is possible to reflect the behavior of every class at every device accurately. Clearly, it is desirable to have the model represent most accurately the behavior of the critical (mostly heavily used) resources. Similarly, if one class of customers is of particular interest in a modelling study, then validation of the model should place special emphasis on the performance measures of that class. Table 12.3 suggests rough guidelines for reasonable expectations of model accuracy during validation.

An important point to note is that queueing network models typically project percentage changes in performance with more accuracy than absolute levels of performance. For example, consider the projection of the effect on interactive response time of adding a batch workload to a system. Assume that the measured response time in the original system was six seconds, and the baseline model validated within 20%, giving a response time of five seconds. If the modified model then projected a ten second response time after the batch workload was added, we should anticipate a response time in the modified system of twelve seconds (rather than ten) since the model projected a doubling of the response time.

model type	**system throughput**	**system response time**	**device utilizations**	**device queue lengths**
single class	0 to 5%	5 to 20%	0 to 5%	5 to 20%
multiple class (per class)	5 to 10%	10 to 30%	5 to 10%	10 to 30%

Table 12.3 – Reasonable Tolerances in Validation

Often, even in well conceived and well executed modelling studies, an initial model will not satisfy the validation criterion. In such cases, reasonable modifications of the assumptions used in estimating input parameters (especially service demands) should be attempted. For example, by noting which classes have throughputs underestimated, the analyst may be guided in a reassessment of how overhead should be attributed to the various classes. This review is repeated until the model can be validated. It is not unusual for several iterations to be required at this stage. In some cases, however, no reasonable technique for estimating inputs yields acceptable results. This is a sign that some important aspect of the system's behavior has not been captured in the model. In many such cases, accuracy can be improved by adding more detail to the model.

It is important to realize the significance of validating a model successfully. If information from measurement data is used to establish values of model inputs, then the fact that the model outputs match the measurement data is, at first glance, not surprising. After a little thought, however, one realizes that success in validation carries the significant implication that the numerous assumptions made in establishing the model are acceptable in the context of the particular system under study. With a validated model, we are prepared to proceed to the modification analysis and performance projection, the subjects of the next chapter.

12.7. Summary

The inputs required by queueing network models can be divided into three groups: the customer description, the center description, and the service demands. The information required to determine the values of these inputs is obtained from a system description and data recorded and reported by various monitors. Many of the input values can be determined in a straightforward manner from this information. Other values, however, must be inferred. The bulk of this chapter has been devoted to techniques for doing so, for various inputs.

An appropriate modelling strategy is to start with the simplest model that might suffice, adding detail as necessary. The process of model validation may involve several iterations in which input values are revised and detail is added.

Thorough validation must be based on several measurement intervals. It also must be based on knowledge of the kinds of performance projection questions for which the model is to be used.

12.8. References

Several good books on computer system performance measurement techniques are available, such as [Ferrari 1978], [Ferrari et al. 1983], and [Svobodova 1976]. These, however, do not deal specifically with the needs of queueing network modelling.

Rose [1978] treats the queueing network parameterization problem in general, and also relates the techniques to various specific systems. Kienzle and Sevcik [1979] review the approaches to parameterization taken by a number of early queueing network modelling case studies.

Curtin [1979] describes a performance database which serves as a repository for measurement data, and which can be accessed by the SAS statistical analysis package to produce reports suitable for both managers and analysts. Lindsay [1980] reports on the accuracy of a software performance monitor by comparing its results to those of a hardware monitor.

Artis [1979] suggests a technique for identifying customer classes based on the similarity of their resource demand patterns. Cooper [1980] describes both the identification of customer classes and the use of capture ratios as part of his presentation of an overall capacity planning methodology. Anderson [1979] proposes a sophisticated method for apportioning unattributed device activity to classes using multiple linear regression.

The details of the parameterization process depend heavily on the system under consideration. Both the quantity and the quality of data varies widely among systems. Consequently, proceedings of "user group" conferences are good sources of papers describing techniques of relevance to a particular type of system.

[Anderson 1979]
Edwin Anderson. A Method for the Estimation of Resource Use for Queueing Models. *Proc. CMG X International Conference* (1979), 157-164.

[Artis 1979]
H. Pat Artis. A Technique for Establishing Resource Limited Job Class Structures. *Proc. CMG X International Conference* (1979), 249-253.

[Cooper 1980]
J.C. Cooper. A Capacity Planning Methodology. *IBM Systems Journal 19*,1 (1980), 28-35.

[Curtin 1979]
James P. Curtin. An MVS Performance Data Base and Reporting System Using SAS. *Proc. CMG X International Conference* (1979), 35-39.

[Ferrari 1978]
Domenico Ferrari. *Computer Systems Performance Evaluation.* Prentice-Hall, 1978.

[Ferrari et al. 1983]
Domenico Ferrari, Giuseppe Serrazi, and Alessandro Zeigner. *Measurement and Tuning of Computer Systems.* Prentice-Hall, 1983.

[Kienzle & Sevcik 1979]
Martin G. Kienzle and Kenneth C. Sevcik. Survey of Analytic Queueing Network Models of Computer Systems. *Proc. ACM SIGMETRICS Conference on Simulation, Measurement and Modeling of Computer Systems* (1979), 113-129.

[Lindsay 1980]
David S. Lindsay. RMF I/O Time Validation. *Proc. CMG XI International Conference* (1980), 112-119.

[Rose 1978]
Clifford A. Rose. A Measurement Procedure for Queueing Network Models of Computer Systems. *Computing Surveys 10*,3 (September 1978), 263-280.

[Svobodova 1976]
Liba Svobodova. *Computer Performance Measurement and Evaluation Methods: Analysis and Applications.* North-Holland, 1976.

12.9. Exercises

1. Section 2.2 describes two case studies in which queueing network models were used for performance projection in an IBM processing complex. In each case, the objectives and the results of the study were presented, but the details of the model were not. For each of these studies, use the available information to specify an appropriate structure for a model. Indicate the significant parameters of the model and suggest how their values might be established.
2. In a system with two workload components, batch and interactive, the following measurements were obtained in a 60 minute observation interval:

   observed CPU busy time: 50 minutes
   accounted batch CPU time: 20 minutes
   accounted interactive CPU time: 10 minutes

   a. Assuming that the "capture ratio" is the same for each workload component, what proportion of the observed CPU busy time should be attributed to each component?
   b. Assuming that the primary source of CPU overhead is page transfers and that 75% of all page transfers are for interactive customers, what proportion of the observed CPU busy time should be attributed to each workload component?
   c. In a second 60 minute observation interval, the observed CPU busy time was 45 minutes, while the accounted CPU times for batch and interactive were 15 and 10 minutes, respectively. Using the measurement data from both observation intervals simultaneously, what proportion of the observed CPU busy time should be attributed to each workload component?
3. In an observation interval, the number of logical I/Os (in thousands) for classes A, B, and C were 60, 50, and 30, respectively. In the same interval the number of physical I/Os (in thousands) at the two disk drives were 100 and 60, respectively. Determine an appropriate allocation to each class of the physical I/Os at each disk drive under each of the following assumptions:
   a. No further information is available.
   b. The ratios of physical to logical I/Os for classes A, B, and C are known to be approximately 13/12, 11/10, and 4/3, respectively.