Chapter 16

Using Queueing Network Modelling Software

16.1. Introduction

A variety of techniques for evaluating queueing network models have been described. The general techniques of bounding analysis, single and multiple class analysis, and hierarchical modelling were presented in Part II. Specific techniques for memory, disk I/O, and processor subsystems were discussed in Part III. This collection of techniques comes together for the computer system performance analyst in the form of queueing network modelling software. Such software frees the analyst from the algorithmic portion of the modelling process, allowing concentration on important issues such as model construction and validation, performance projection, and capacity planning.

Most queueing network modelling software can be understood in terms of the structure illustrated in Figure 16.1. There is a sequence of software layers, each transforming input received from the layer above into output suitable for the layer below. In Section 16.2 we will refer to this structure in describing the major components of queueing network modelling software. In Section 16.3 we give an example of conducting a performance study using such software.

16.2. Components of Queueing Network Modelling Software

16.2.1. The Core Computational Routine

The job of the *core computational routine*, situated at the lowest level in Figure 16.1, is simply stated. Given a separable queueing network model defined by its customer description, center description, and service demands, this routine produces performance measures. In other words, the core computational routine is an embodiment of the techniques described in Part II of the book. The core routine may be based on either exact or approximate algorithms.

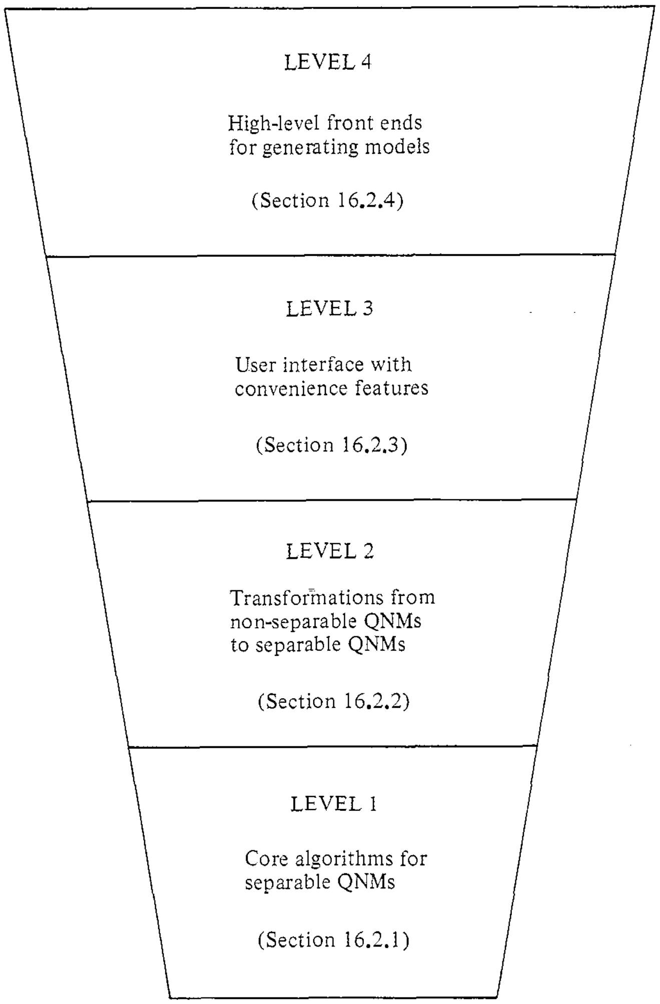
LEVEL 4
High-level front ends for generating models
(Section 16.2.4)
LEVEL 3
User interface with convenience features
(Section 16.2.3)
LEVEL 2
Transformations from non-separable QNMs to separable QNMs
(Section 16.2.2)
LEVEL 1
Core algorithms for separable QNMs
(Section 16.2.1)

Figure 16.1 – The Structure of Queueing Network Modelling Software

16.2.2. The Approximation Transformations

When viewed at a detailed level, many subsystems have characteristics that lead to non-separable queueing network models. The software layers immediately above the core computational routine, the *approximation transformations* (level 2 in Figure 16.1), translate these non-separable models into a form that is suitable for the core routine. In other words, the approximation transformations correspond to the techniques described in Part III. Many of these techniques require an iterative relationship with the core routine: the transformations provide input suitable to the core routine, and the core routine's output is used as additional input to the transformations.

Consider the treatment of C classes with independent memory constraints in Section 9.3.2. In a sense, there are C different separable low-level models, and C different separable high-level models. The low-level model corresponding to class c has each class other than c represented by its average central subsystem population, while class c's population is varied from 1 to M_c, its memory constraint. The high-level model corresponding to class c is a single class model using the FESC obtained from the low-level model. Each model is evaluated by the core routine, with the transformation layer using the outputs of some models to define the inputs of others.

Another example is the treatment of RPS disks in Section 10.3. The service demands at the centers representing the disks must reflect more than the seek, latency, and data transfer requirements. Approximation transformations estimate the component due to path contention, as in Algorithm 10.2, and calculate *effective* service demands, which are passed to the core routine. Again, this process is iterative.

Additional examples of transformations include those used to represent priority scheduling (Section 11.3) and distributions of multiprogramming level (Section 9.2).

16.2.3. The User Interface

An important attribute of queueing network modelling software is the convenient expression of performance models. The *user interface*, level 3 in Figure 16.1, must bridge the gap between the world of queueing network models and the world of computer systems, so that performance studies can be conducted efficiently by analysts whose primary training is in computer systems.

In many cases, queueing network concepts correspond directly to computer system concepts: e.g., classes to workload components, centers to devices, customers to users or jobs. These queueing network concepts are made visible to the analyst with little intervention by the user interface layer.

In other cases, the correspondence is not so direct. In particular, the user interface layer smoothes the analyst's interaction with the transformation layer. As an example, when multipathing I/O is modelled, as described in Section 10.5, the user interface layer allows the analyst to describe the system in terms of channels, controllers, heads of string, disks, and logical channels (in the case of an IBM configuration) and translates this information into a form acceptable to the transformation layer. The transformation layer then uses this information to estimate effective service demands for the various disks, interacting iteratively with the core routine, which evaluates a sequence of separable models provided by the transformation layer. Finally, the transformation and user interface layers provide performance measures in a meaningful form.

A related facility allows the analyst to associate a "type" with a device, and to specify the relevant characteristics of this type to the software. This is useful during modification analysis. As a simple example, an existing system may have an IBM 3081-D CPU, and one of the contemplated modifications may be an upgrade to a 3081-K. If the relative speeds of these two processors are known to the software (in fact, they are roughly 4.0 and 5.5), then the following sequence of interactions would be possible:

- analyst provides all inputs for baseline model
- analyst identifies CPU as 3081-D
- baseline model is validated
- analyst specifies that CPU type should be changed to 3081-K
- software adjusts CPU service demands based on internal information

A similar approach can be applied to other devices.

The user interface layer typically provides the ability to save, recall, and edit model definitions during an interactive session, since model modification is the major activity in interacting with queueing network modelling software. Output reports also need to be stored for post-processing.

A final example of a facility provided by the user interface layer is a means for the analyst to "program" the package using a simple language, similar to the "exec file" or "command file" facility provided by most contemporary timesharing systems. A simple application of this facility would be to perform automatically a parametric study (for example, on the effect of increasing the number of active terminals). Of course, an analyst could conduct such a study by interacting with the software directly, issuing separate commands to evaluate the model with each population of interest. A better approach, though, would be to write a simple program that, when interpreted by the package, accomplishes this task. A more sophisticated application would be to implement new specialized evaluation techniques for subsystems peculiar to a particular environment, similar to the established techniques described in Part III.

16.2.4. High-Level Front Ends

Given measurement data for a system of realistic size, it takes a significant amount of time to calculate and enter the inputs of a queueing network model. The process is error-prone, because of the large volume of data. Many of the actions are repetitive. All of these factors argue strongly in favor of a computer program that partially automates the construction of baseline models of existing systems. This is one major example of a *high-level front end*, level 4 in Figure 16.1. Other examples of application areas in which high-level front ends are of great utility include performance projection for proposed systems, for database systems, and for communication networks. We touch upon each of these in turn.

Modelling Existing Systems

In Chapter 12 we discussed the parameterization of baseline models of existing systems. To be sure, parts of this process require subtle judgements on the part of the analyst. For any particular system, though, it is possible to automate a large proportion of the labor involved in translating measurement data (stored in a specified format in a performance database) into queueing network model inputs.

The feasibility of constructing such a high-level front end relies on the fact that the general structure of queueing network models, while system dependent, is not highly installation dependent. For example, in constructing queueing network models of IBM MVS systems it is reasonable to equate "performance groups" with customer classes. In general, such front ends construct fairly simple models, which subsequently are adjusted by the analyst. Overall, the savings in time can be significant.

Modelling Proposed Systems

Performance projection for proposed systems was discussed in some detail in Chapter 14, where we illustrated the syntax of two high-level front ends. The interface for this application is based on the system designer's point of view. It deals with the natural units of the application (e.g., estimated number of disk reads on the software side, transfer rate on the hardware side) and translates them into a form acceptable to the core routine of the queueing network modelling software.

Modelling Database Systems

In systems where database processing has a major influence on performance, the use of a specialized high-level front end can expand the scope of performance questions that can be addressed conveniently.

The presence of a significant database workload component does not detract from the applicability of the techniques described in Parts II and III to the problem of projecting performance. In fact, several of the case studies described earlier treat systems in which database activity is significant. There are performance questions that arise in the context of database systems, however, that cannot be addressed conveniently using the interface provided by general queueing network modelling software.

The efficient operation of database systems depends on many design decisions, including:

- representation of the logical data model as a set of files
- specification of links that relate records to one another
- selection of indices to facilitate access to records having various values for a certain attribute in a file
- choice of query processing strategy
- placement of indices and files in main memory and on various storage devices
- allocation of buffer space for various purposes

All of these decisions have a substantial influence on the service demands of database transactions. The purpose of a high-level front end for database systems is to support investigation of decisions such as these.

In most database systems, a few transaction types are dominant. If we represent each such transaction type as a customer class in a model, the class can be characterized by its workload intensity and its typical pattern of accesses to items in the database. The front end can calculate the number of physical block accesses per transaction from the pattern of accesses to logical data items, by taking into account the representation of the logical database as files and links between them. In processing a query, the order in which operations are done (the query processing strategy) and the presence of indices on selected attributes also influence the number of block accesses. Finally, data placement decisions determine the fraction of block accesses directed to each device, and buffer strategies determine the fraction of block accesses that require a physical data transfer. Thus, by considering the characteristics of a database environment and the design decisions made, the front end can transform the high-level specification of transaction types into a specification suitable for input to a standard queueing network modelling package.

Specific database management systems impose specific restrictions. For example, IMS and System 2000 support particular ways of linking and indexing files, and provide particular tuning parameters (such as priority specifications, and the allocation of buffer space to various uses). A high-level front end tailored to a specific database system is most convenient for an analyst because all the decisions resolved within the system can be built into the front end.

Modelling Computer Communication Networks

Many issues need to be considered in the design of computer communication networks: network bandwidth, multiplexing and concentration strategies, network protocol layers, flow control policies, routing strategies, and buffering strategies. A network designer is interested in knowing if a proposed network can handle a projected workload intensity while providing an acceptable level of performance. A front end for this application accepts system descriptions in terms of entities such as cluster controllers, line speeds, host/satellite topology, and protocols. These are transformed for the queueing network model into service demands at various centers (e.g., the communications controller). Hierarchical modelling, as described in Chapter 8, is useful here.

16.3. An Example

This section gives an example of the way in which advanced queueing network modelling software can be used by an analyst to develop and modify a model of a large contemporary computer system. We have three objectives:

- We wish to illustrate the levels of detail that are appropriate for building a model and using that model for performance projection.
- We wish to illustrate the relationship between modelling concepts, evaluation algorithms, and modelling software.
- We hope to indicate how such software can increase the productivity of the analyst.

To achieve these objectives, it is necessary to include example commands for specific queueing network modelling software. We have chosen the package MAP for our example. Other packages have similar features.

16.3.1. Description of the Example System

The system we treat is an Amdahl 470 V/7 with 16 million bytes of main memory. It runs IBM's MVS operating system with two important workload components, batch and TSO (interactive). Other workload components are present, but the performance questions of interest concern only batch throughput and TSO response time.

Approximately 200 disk drives, IBM 3350s, are accessible through eight physical channels. Other devices are attached to the system (e.g., unit record devices and tape devices), but they have little influence over performance in the current system or any contemplated future system.

16.3.2. Building a Baseline Model

The first step in our modelling process is to construct a baseline model that validates well on batch throughput and TSO response time. To present the concepts clearly, we will be showing how an analyst might interact directly with the package in building the baseline model, rather than using a high-level front end to partially automate the process, as described in Section 16.2.4. As a practical consideration, though, some software assistance is a necessity in building large models. We assume that all necessary parameter values have been obtained from measurement data in accordance with the procedures suggested in Chapters 12 and 17.

Our discussion of model building treats classes, centers, domains, and service demands in turn.

Classes

Because of the performance questions of interest, the batch and TSO workload components must be represented as separate classes. Multiple class algorithms similar to those of Chapter 7 will provide the necessary class based performance measures. In the actual system, other workload components interfere with these two important components. We must include this effect in the model. We do so with an aggregated artificial third class. (Neglecting these other components would yield optimistic results for the classes of interest.)

The definition of a class includes its name, its type, and its workload intensity. TSO is a class of TERMINAL type with an associated number of terminals and average think time. PRODUCTION is a class of BATCH type with an associated average multiprogramming level. The artificial class OTHER is specified as a class of TRANSACTION type with an associated arrival rate, set equal to the rate at which jobs of the other workload components complete. (This approach guarantees that the modelled throughput of the OTHER class will equal the aggregated throughputs of the other workload components.)

MAP commands to define the classes are:

```
CLASS TSO              (create class TSO)
TYPE TERMINAL          (state its type)
ONLINE_USERS 68        (specify the number of active terminal users)
THINK 12.53            (think time of 12.53 seconds)

CLASS PRODUCTION       (create class PRODUCTION)
TYPE BATCH             (state its type)
AVG_MPL 8              (set its average multiprogramming level)

CLASS OTHER            (create class OTHER)
TYPE TRANSACTION       (state its type)
ARRIVAL_RATE .11       (arrival rate is .11 jobs/sec.)
```

Additional commands that further specify class attributes will be given shortly, when other necessary components of the model have been defined.

Centers

The model includes a center representing the CPU and a center representing each disk drive. The definition of a center includes its name and optional attributes such as its scheduling discipline and its manufacturer's model designation. The MAP commands to define the CPU and I/O devices are:

```
CENTER CPU             (create a center named CPU)
SCHEDULE PRIORITY      (specify priority scheduling)
MODEL V/7              (inform MAP that it is an Amdahl V/7 CPU)

CENTER SYS001          (create center SYS001)
MODEL 3350             (inform MAP that is an IBM 3350 disk)

CENTER PAG001          (create center PAG001)
MODEL 3350             (it is an IBM 3350)
 .
 .
 .
```

If no scheduling discipline is specified, processor sharing is assumed at a CPU center, and FCFS is assumed at other centers. Here, the CPU has been made a priority center. The relative scheduling priorities of classes at this center must be given. In MAP, this is done using the PLEVEL command:

```
CLASS TSO PLEVEL 3              (TSO has highest priority)
CLASS PRODUCTION PLEVEL 2       (PRODUCTION has middle)
                                (  priority)
CLASS OTHER PLEVEL 1            (OTHER has lowest priority)
```

Higher PLEVEL values indicate higher scheduling priorities. The inclusion of priority scheduling will cause MAP to evaluate the model using a technique similar to the one described in Section 11.3.

Domains

In the actual system, the TSO workload component is subject to a memory constraint of nine processing regions, meaning that at most nine TSO users can be competing for CPU and I/O service at once (other users must queue for memory). In MAP, this is modelled by the DOMAIN feature. The definition of a domain consists of its name and its capacity. It also is necessary to indicate which classes are constrained by each domain. The MAP commands to do this for our example are:

```
DOMAIN DOM_TSO CAPACITY 9       (create domain DOM_TSO;)
                                (  set its capacity to 9 jobs)
CLASS TSO INDOMAIN DOM_TSO      (associate TSO with domain)
                                (  DOM_TSO)
```

The other classes are not associated with domains. The use of the DOMAIN feature will cause MAP to evaluate the model using a technique similar to the one described in Section 9.3.

A final memory related command is MEMSIZE, which informs MAP of the amount of main storage in the base configuration:

```
MEMSIZE 16    (16MB of main storage in the base system)
```

This information is used by MAP during modification analysis, as we will see.

Service Demands

The final components we define in the baseline model are the service demands of all classes at all centers. A convenient way of entering the service demand values is to have the package prompt for them in a systematic manner. In MAP, this is accomplished by specifying class and center values of ALL. Then, in response to the DEMAND command, MAP will print class and center names and accept the corresponding service demands, as in:

```
CLASS  ALL      (ALL will cause MAP to prompt with all class names)
CENTER  ALL     (ALL will cause MAP to prompt with all center)
                (  names)

DEMAND          (indicate to MAP that we want to specify service)
                (  demands)

TSO:            (MAP prompt for TSO, to which the following)
                (  prompts apply)
   CPU:         (MAP prompt for CPU service demand)
      .09       (user-specified)
   SYS001:      (MAP prompt for SYS001 service demand)
      .04       (user-specified)
     .
     .
     .
```

(Prompts printed by MAP are shown in italics.)

Performance Reports

Having defined the model, a number of performance reports can be obtained for the baseline system. Examples are shown in Table 16.1.

System Performance Measures

Class	Response Time	Time in System	Memory Wait	Throughput
TSO	5.6423	2.3868	3.2555	3.7420
PRODUCTION	12.5391	12.5391	0.0000	0.6380
OTHER	7.2880	7.2880	0.0000	0.1100

Device Utilizations

Center	TSO	PRODUCTION	OTHER	Total
CPU	0.3368	0.5104	0.0220	0.8692
SYS001	0.1497	0.1276	0.0033	0.2806
PAG001	0.1123	0.2233	0.0330	0.3686
⋮				

Table 16.1 – Example Performance Report

The model can be validated by checking its calculated performance measures against those from several measurement periods, as described in Chapters 2 and 12. This often is an iterative process in which the model and its parameters are refined as a better understanding of the system is gained.

At this point, we assume that the model has been validated successfully. Once this has been accomplished, the model definition can be saved:

```
SAVE BASELINE    (save model definition in permanent file)
```

The model could be retrieved subsequently, either in this MAP session or in other sessions, using:

```
READ BASELINE    (read the named model definition file)
```

16.3.3. Reflecting Anticipated Changes

In Chapter 13 we discussed the parameterization of models of evolving systems. Many model modifications are possible. In this subsection we will discuss modifications to the workload, hardware, and software components of the baseline model. In doing so, we will illustrate many of the convenience features of contemporary queueing network modelling software.

Workload

One standard workload change is an increase in workload intensities. To specify a new value for the PRODUCTION multiprogramming level, the AVG_MPL command would be issued with a new value as its operand:

```
CLASS PRODUCTION AVG_MPL 12    (set the new value)
```

A more usual specification involves a relative change, e.g., a 20% increase in the TSO workload intensity (number of terminals), as in:

```
CLASS TSO ONLINE 1.2*    (multiply the previous value by 1.2)
```

(Any unique prefix is an acceptable abbreviation in MAP; ONLINE is a shortened form of ONLINE_USERS.)

Other workload changes might involve the service demands. New application software might reduce the CPU path lengths for the TSO class. This might be reflected in the model by:

```
CLASS TSO CENTER CPU DEMAND .90*    (CPU demand is)
                                    ( reduced by 10%)
```

The service demands at the disks might change as well. New blocking strategies for files might reduce the number of I/Os per transaction, resulting in reduced access time per byte transferred because of smaller total seek and latency requirements. Disk service demands can be modified in a manner similar to above.

Having modified the parameters of the model as appropriate, performance estimates for the proposed system can be obtained using the PERFORMANCE command, as was done during the validation phase of the study.

Hardware

A typical hardware change is the upgrading of a device to a more powerful one. In our example, the V/7 CPU might be upgraded to a 5860. Because a MODEL has been specified for the CPU center, specifying a new MODEL has the effect of changing the service demands of all classes at the CPU:

```
CENTER CPU MODEL 5860    (represent upgrade to Amdahl 5860)
```

This change is done automatically within the package based on built-in knowledge of the relative speeds of these two CPUs.

Just as a CPU can be upgraded, so can disks. If some or all of the 3350 disks are changed to 3380 disks, the MODEL command can be used to alter the service demands of all classes at the devices upgraded, as in:

```
CENTER SYS001 MODEL 3380    (upgrade from IBM 3350 to)
                            ( IBM 3380)
```

Another typical upgrade involves a memory expansion. Going from 16 megabytes to 24 megabytes allows additional space to be allocated to the user workloads. How this space is allocated is dependent on the operating system memory policies. MAP uses built-in knowledge to estimate how a memory expansion will affect various classes. The command:

```
MEMSIZE 24    (increase main memory size to 24MB)
```

causes MAP to alter automatically the average multiprogramming level of the PRODUCTION class and the domain capacity of the TSO class to reflect the use of increased main memory. (The values computed by MAP are estimates and account only for first-order effects; the analyst might want to modify them on the basis of more detailed knowledge.)

As a final hardware change, consider the introduction of an additional controller to the I/O subsystem to reduce path contention. (See Chapter 10 for a more complete description of the I/O subsystem architecture being considered.) Our basic model contains no explicit representation of I/O paths (the effect of contention for I/O paths is reflected in the disk service demands, which contain a path contention component), so to model this change a more detailed representation of the I/O subsystem is required. We illustrate the process of creating a detailed model of this sort with an example. For purposes of exposition, we will keep the example small.

Suppose two strings of disks can connect to memory through two controllers, and these two controllers can connect through two channels. This connection scheme comprises what we call a *logical channel*, and must be represented in our detailed model. Before defining the logical channel in MAP, the basic components should be created, as in:

```
CHANNEL  CH1           (define a channel)
CHANNEL  CH2           (define another channel)

CONTROLLER  CTLA       (define a controller)
CONTROLLER  CTLB       (define another controller)
```

The logical channel now can be defined:

```
LCHANNEL  LCH      (define the logical channel; MAP)
                   (  now will prompt for path descriptions)
   CHANNEL:        (MAP prompt for channel at head of this path)
     CH1           (user-specified)
   CONTROLLER:     (prompt for controller on this path)
     CTLA          (user)
   CONTROLLER:     (prompt for another controller on this path)
     CTLB          (user)
   CONTROLLER:     (prompt)
                   (user null line indicates end controller list)
   CHANNEL:        (prompt for another channel to head)
     CH2           (  another set of paths)
   CONTROLLER:
     CTLA
   CONTROLLER:
     CTLB
   CONTROLLER:
                   (user null line, to end controller list)
   CHANNEL:
                   (user null line, to end channel list)
```

Disks are grouped into sets called strings, with all disks on the same string accessible over the same set of I/O paths (i.e., the same logical channel). At this point, strings and associated information must be defined, as in:

```
STRING  STR1                        (create a string)
ONLCHANNEL  LCH                     (inform MAP of the paths by)
                                    (  which disks on the string)
                                    (  are accessible)
STRING  STR2                        (create another string)
ONLCHANNEL  LCH                     (place STR2 on paths LCH)

CENTER  SYS001  ONSTRING  STR1      (put SYS001 on string STR1)
  .
  .
  .
```

To model the addition of a controller to the system, the logical channels affected by the new controller could be redefined to include it, and the model re-evaluated. MAP automatically estimates the new level of path contention, and uses this to alter the disk service demands.

Other changes, such as the addition of channels and strings or the movement of disks among strings, can be modelled similarly.

Software

By software changes we mean changes in the operating policies or parameters of the system, not changes in the intrinsic workload demand. As an example, consider attempting to balance resource usage by moving a set of TSO files from one disk, where these files are responsible for one third of the accesses, to another. This can be represented in MAP using:

```
CLASS TSO  MOVE  .33  SYS001  SYSTMP
```

We can change the priority scheduling structure at the CPU simply by specifying new PLEVEL values, as in:

```
CLASS  TSO  PLEVEL  2               (reduce TSO's priority level)

CLASS  PRODUCTION  PLEVEL  3        (give PRODUCTION priority)
                                    (  over TSO)
```

PRODUCTION now has priority over TSO at the CPU.

Unlimited variations are possible, but the essence of constructing and modifying a model should be evident. The interactions between user and package that have been illustrated are typical of those involved in performance projection studies. The usual goal of such studies is to estimate the performance of an existing system subjected to new workloads and configurations. To support this activity, the software allows the analyst to modify classes, centers, domains, and service demands in a simple manner. The ability of the software to represent device-specific and system-specific information is especially advantageous. A wide range of alternatives can be investigated rapidly and interactively.

16.4. Summary

Queueing network modelling software can be viewed as consisting of four levels. From lowest to highest, they are:

1. The *core computational routine*, which evaluates separable queueing network models as described in Part II.
2. The *approximation transformations*, which interact with the core routines to evaluate detailed, non-separable models of subsystems such as memory, disk I/O, and processors, as described in Part III.
3. The *user interface*, which allows the analyst to use the terminology of computer systems, rather than the terminology of queueing networks, and which also supports facilities such as the filing and retrieval of model definitions and output reports, and the programmability of the software.
4. *High-level front ends*, which partially automate specific tasks described in Part IV: producing queueing network model inputs from system measurement data, iteratively evolving the system specifications needed for projecting the performance of proposed systems, etc.

All of these levels need not be present; indeed, simple queueing network modelling software often consists only of the first level. However, the higher levels are important to the professional computer system performance analyst. The four levels need not be packaged together in a single piece of software; it is typical for the fourth level to be separate from the other three.

An obvious question that arises is whether queueing network modelling software should be developed in-house, using information from sources such as this book, or obtained from a vendor. Most of the arguments support the latter choice. Many of these arguments are managerial, but one is technical, and we will consider it briefly.

Queueing network modelling technology has advanced rapidly in the recent past, and can be expected to continue to do so in the near future. That portion of a computer system performance analyst's time not devoted to computer system analysis is better spent staying abreast of advances in computer systems than staying abreast of advances in queueing network technology.

A quick historical review in support of this point may be of interest. Table 16.2 shows that even at the relatively well understood level of the core computational routine, advances have been recent, rapid, and significant. At the level of the approximation transformations, progress has been even more recent. For example, the techniques for evaluating multiple class memory constrained queueing networks (Section 9.3) and for evaluating multipathing I/O subsystems (Section 10.5) both were developed during the two year gestation period of this book. In other words, extensive changes have taken place recently even in the algorithms at the lower levels of queueing network modelling software.

rough date	**development**
1965	First application of queueing network models to computer systems: a two center model with a population of a few customers, evaluated using the global balance technique.
1970	First efficient evaluation algorithm, an exact technique (the "convolution method") for single class separable models.
1975	Extension to multiple class separable models.
	Concept of flow equivalent service centers using load dependent servers.
1980	Mean value analysis and, subsequently, the highly efficient MVA-based iterative approximate evaluation techniques for separable models.

Table 16.2 — Advances in the Core Computational Routine

16.5. Epilogue

Given that much of what has been discussed in this book can be — and has been — packaged in queueing network modelling software, why have you and we together labored so long over this material? The reason is simple: the effectiveness with which such software can be applied is multiplied many times by an understanding of the principles and techniques upon which it is based. Briefly:

- In the case of Part I, you learned that Little's law and its relatives, which provide the technical foundation of queueing network modelling, are reasonable, and are of extremely broad applicability. This knowledge, along with an awareness of the widespread success of performance studies using queueing network models, provides confidence in the approach.

- In the case of Part II, you learned the techniques used to evaluate separable queueing network models, and the assumptions upon which these techniques rely. The robustness of these techniques with respect to the assumptions was indicated. This knowledge again builds confidence in the approach and, more importantly, it indicates the range of applicability of queueing network models, and provides insight into the ways in which detailed models of specific subsystems can be constructed using separable queueing network models as a basis.
- In the case of Part III, you learned a collection of techniques for augmenting the algorithms of Part II to evaluate detailed models of specific subsystems, where it often is necessary to represent the effects of system characteristics that violate the separability assumptions. This knowledge will help you understand the homogeneity assumptions made by queueing network modelling packages in specific cases (for example, in representing multipathing I/O subsystems), so that you will know if these assumptions should be a source of concern in a particular performance study. The techniques presented in Part III also can serve as a model for similar techniques that you might devise yourself when confronted with a unique modelling problem.
- In the case of Part IV, you learned how to parameterize queueing network models to conduct studies of existing, evolving, and proposed systems. Often a data reduction tool will not be available, and you will be forced to work from raw measurement data. Even if such a tool is available, it often will be desirable to augment it to accommodate the requirements of a particular installation. You now have the knowledge to do so.
- In the case of Part V, you learned through example how queueing network models can be applied in "non-traditional" contexts. As computer systems continue to evolve, it is important to recognize that the applicability of queueing network technology, and of existing queueing network modelling software, extends well beyond the confines of centralized systems with simple characteristics.

To repeat some comments made in the Preface, queueing network models, while not a panacea, are the appropriate tool in a wide variety of applications. Computer system analysis using queueing network models is a blend of art and science, requiring both education and experience. We hope to have contributed, and wish you success in applying queueing network models in your work.

16.6. References

Our discussion of the structure of queueing network modelling software is based on the treatment by Graham, Lazowska, and Sevcik [1982]; this paper is the source of Figure 16.1.

BEST/1, available since the late 1970s and considerably updated since then, is one of the earliest commercial queueing network modelling packages [BGS 1982a]. More recent packages include CMF/Model [Boole & Babbage 1983], RESQ [Sauer et al. 1982], and MAP [QSP 1982a, 1982b].

An example of a high-level front end for modelling existing systems is CAPTURE/MVS [BGS 1982b], which prepares BEST/1 input from MVS performance monitor data.

A view of how to structure a hierarchical tool for database performance projection, including a sequence of database workload descriptions, is provided by Sevcik [1981]. A front end interface for projecting performance of System 2000 databases is described by Casas Raposo [1981].

[BGS 1982a]
BEST/1 User's Guide. BGS Systems, Inc., Waltham, MA, 1982.

[BGS 1982b]
CAPTURE/MVS User's Guide. BGS Systems, Inc., Waltham, MA, 1982.

[Boole & Babbage 1983]
CMF/Model. Boole & Babbage, Inc., Sunnyvale, CA, 1983.

[Casas Raposo 1981]
I. Casas Raposo. Analytic Modeling of Data Base Systems: The Design of a System 2000 Performance Predictor. Technical Note 25, Computer Systems Research Group, University of Toronto, July 1981.

[Graham et al. 1982]
G.S. Graham, E.D. Lazowska, and K.C. Sevcik. Components of Software Packages for the Solution of Queueing Network Models. *Proc. CPEUG '82* (1982), 183-187.

[QSP 1982a]
MAP User Guide. Quantitative System Performance, Inc., Seattle, WA, 1982.

[QSP 1982b]
MAP Reference Guide. Quantitative System Performance, Inc., Seattle, WA, 1982.

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

[Sevcik 1981]
K.C. Sevcik. Data Base System Performance Prediction Using an Analytical Model. *Proc. 7th VLDB Conference* (1981), 182-189.