This paper provides a nontechnical introduction to queuing network concepts, a short introduction to results of research into analytical modeling methodology, and a set of simple but typical examples that illustrate the application of that methodology to problems of computer performance.

Interactive modeling of computer systems

by M. Reiser

The objectives of this paper are to introduce concepts of the theory of queuing networks, to describe an APL program—called QNET4—for the solution of queuing network problems, and to give examples of the analysis of computer systems using the methods of QNET4. In the process of constructing models, we wish to clearly distinguish between modeling and methods of solution.

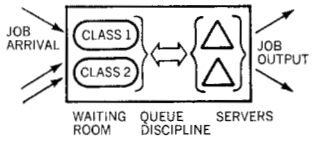
Modeling is the process of mapping a real system into a suitable abstract representation, such as an equation or a queuing diagram. It is the modeler who decides those aspects of a system that are sufficiently important to be represented. Modeling, especially on a high level of abstraction, is a difficult endeavor. Much remains to be done to establish a proven modeling methodology.

Methods of solutions are necessary to provide numerical results. A method of solution is usually capable of solving a certain class of models. (Such a class of models is sometimes called an *abstract model.*) The abstract model of this paper is the network of queues. Algebraic and simulation methods are two important methods of solution of such models. The APL program discussed in this paper, which we call QNET4, is an implementation of the algebraic method of solution. Note that we view simulation as a method of solution, not as a model class that is distinct from the class of algebraic (or analytic) models.

Queues and queuing networks

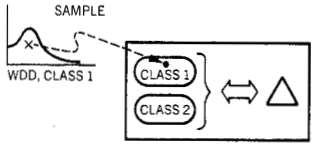
The term *queue* denotes an individual resource of the queuing network. Several queues are connected into a queuing network by "routing," which is described later in this paper. A queue consists of the following three elements: a waiting room, a set of servers or processors, and a queue discipline, i.e., a rule for determining the scheduling of service to jobs.

Figure 1 A queue with two servers and a waiting room that is organized into two classes



A diagram of a queue is provided in Figure 1. The waiting room of a queue may be subdivided into a set of local classes, each of which may have its own workload characteristics (workload classes). Classes such as priority may also be used by the queue discipline. To further specify a queue, assume a service mechanism by which jobs arrive as members of one of the queue's local classes. (Multiple arrivals are to be excluded.) Upon arrival, a job places a *work demand* (or service demand) on the queue. The work demand is a random variable that is generated by a stochastic process. The method to be discussed here requires that a process be a renewal process, which is to say that successive work demands are independent samples drawn from a distribution, called the Work Demand Distribution (WDD), as illustrated by Figure 2.

Figure 2 A work demand is assigned to a newly arrived Class 1 job



Note that each class may have its own WDD. Work demands are measured in such units as numbers of instructions (path lengths), or numbers of bytes to be transmitted (record size), or other such quantities. Servers provide work at a given speed, called *work rate* or *service rate*. Examples of rate units are Millions of Instructions Per Second (MIPS), thousands of bytes per second (KBPS), baud or rate of telegraphic transmission, or other such quantities. We allow the rate to be a function of the queue size.

Jobs are scheduled to receive service by a rule called *queue discipline*, and stay at the queue until all work is done. Completed jobs depart instantly. The queue discipline may subdivide the work into several slices (e.g., round robin scheduling). In the case of multiple servers, the queue discipline also assigns servers. The queue discipline may make use of the local classes (e.g., priority), and/or the discipline may impose an ordering on the jobs in the waiting room (e.g., order of arrival).

Our definition of the service mechanism differs somewhat from the standard queuing literature where *service times* and their distributions are more customary. In the simple case of constant work rate C and First Come First Served (FCFS), Last Come First Served (LCFS), Priority (PR), or Infinite Servers (IS) queue discipline, service times τ, and work demands m are simply related by the following expression:

$$C\tau = m \tag{1}$$

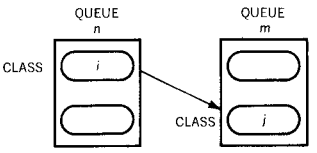
A large literature exists on the single-queue solution, which is available under quite general assumptions; an excellent introduction is given in Reference 1.

network of queues

A network of queues is a multiresource system that has the following characteristics:

- A set of queues—as just described—each having its own distinct service mechanism.
- An arrival process that describes job arrival at the network (open network); or a fixed set of jobs that stays indefinitely in the network (closed network).
- A rule, called *routing*, that governs the way jobs proceed through the network.

Figure 3 Routing transition $i \rightarrow j$

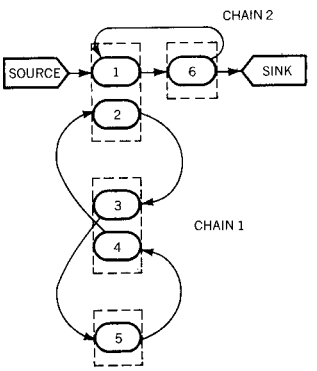


It is often convenient to view routing as a sequence of decisions that are made upon arrival and after each service completion. For example, a routing may specify that a class i job after service completion at queue n may join queue m as a class j job. Schematically, such a transition may be depicted by the arrow as shown in Figure 3.

If all classes have distinct names, queue names need not be given to uniquely specify a transition. The transition of Figure 3 is designated by the following notation:

$$i \rightarrow j \tag{2}$$

Figure 4 Example of a routing graph



Here the rectangular boxes symbolize queues, and the ovals are classes within the queues. Routing imposes a structure on the queuing system that may be represented by a directed graph, an example of which is given in Figure 4. The routing may define classes of nodes, such that no transitions can occur from one class into another. Such classes are termed *chains*. Figure 4 shows a routing with two chains, one of which is open and the other closed. In Figure 4, nodes are local classes, and edges show feasible transitions. Note the use of local classes to obtain the figure-8 pattern in chain 1.

Without loss of generality, we can assume that routing transitions are instantaneous. A routing rule is said to be *state independent* if the sequential decisions are not influenced by queue sizes or service processes. The term *stochastic routing* or probabilistic routing refers to a state-independent routing rule by which, after each service completion, a successor class is selected at random.

Denote a stochastic routing transition formally as follows:

$$i \rightarrow j; p \tag{3}$$

where p is the probability that class j is chosen as successor of class i. As a consequence of the way in which a network of queues is defined, the event that a job is simultaneously at more than one queue is prohibited.

separable queuing networks

The queuing network model, as previously discussed, is a very general one, and simulation is the principal method for its solution. Thus, further restrictions must be imposed to make the

problem yield to a mathematical analysis. In this section, a class of queuing networks, called *separable queuing networks*, is defined and discussed. Such networks are also termed locally balanced networks or networks that have a product-form solution.

A first significant simplification becomes possible by making the state space enumerable, by restricting the class of distributions to those that can be realized by exponential stages.[2] This class of queuing systems is sometimes called *Markovian queuing networks*. The stationary queue size distribution of a Markovian queuing system, if it exists, is given by a system of homogeneous linear equations, called *balance equations*. This, it seems, is familiar mathematical ground, well suited for the digital computer. Unfortunately the number of equations is either infinite (open networks) or grows combinatorially with the size of the network (closed networks). Thus the general class of Markovian networks is severely computationally limited.

Since the solution of the single-queue problem is available under rather general assumptions, such as the GI/G/1 queue—a single resource system with a general independent arrival process, general service process, and one server—one might expect that a network solution could be synthethized from such general individual queue solutions. This decomposition approach does not work in general, however, since the superposition of output processes of queues that feed another queue is far more general than the renewal input process on which the GI/G/1 solution is based. Some approximate methods that ignore the structure of the sum process have been suggested.[3]

The only way to obtain a general class of queuing network models that is not computationally limited is to impose further restrictions, such that the solution of the balance equations can be obtained a priori. Such networks are called *separable* because the balance equations can be solved by separation of variables techniques. A rather large class of separable networks is discussed in References 4–6, and that class of queue networks is now outlined.

- There are N queues labeled $1, 2, \cdots, N$ that are organized into M classes, $1, 2, \cdots, M$.
- Jobs proceed through the queues, according to a stochastic routing rule, by means of which the routing may be decomposable into chains that are either open and driven by Poisson streams or closed with a fixed population.
- A queue has one or several identical servers that are specified by their work rate, which may be a function of the local queue size.
- The following queue disciplines and Work Demand Distributions (WDD) are admissible:

A. A First Come First Served (FCFS) queue where all job classes share the same exponential WDD (homogeneous workload) is illustrated in Figure 5.
B. A Preemptive Resume Last Come First Served (LCFSPR) with general, class-dependent WDDs is illustrated in Figure 6.
C. A Processor Shared (PS) server with general class dependent WDDs serves all jobs at the queue simultaneously with an individual rate that is inversely proportional to the queue size. Figure 7 illustrates a processor-shared queue in which three jobs are executing simultaneously, each one with an individual rate of $C/3$. Note that the work demands are sampled from two different distributions. If a new job arrives, each job executes at an individual rate of $C/4$. The processor-shared queue discipline may be viewed as a limiting case of round-robin scheduling, where the quantum size shrinks to zero.
D. No queuing with class-dependent general delay time distribution is a queue that is also called Infinite Servers and is abbreviated as IS. In such a queue, a job is held for a random amount of time, regardless of congestion.

Figure 5 A first come first served queue with an exponential work demand distribution

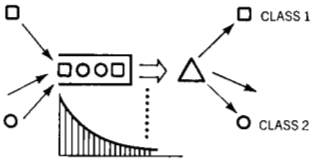
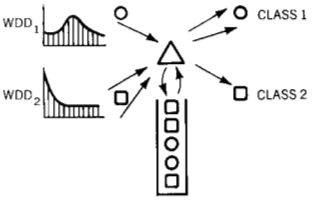


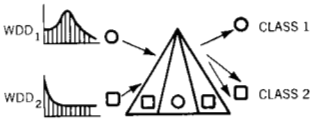
Figure 6 A last come first served queue with a preemptive resume discipline



It is a particularly interesting feature of the solution of separable networks that only the mean work demand enters into the solution. Thus, in those cases where a general distribution is allowed, the form of this distribution is irrelevant. We call such queue disciplines robust; LCFSPR, PS, and IS are robust, whereas FCFS is not. We shall show later in this paper that robustness is very important in practice. The theory of separable queuing networks allows essentially the evaluation of the joint queue size distribution. The following are performance measures that can be obtained:

- Marginal queue size distribution.
- Mean queue size.
- Utilization.
- Average chain population.
- Throughput.
- Average response times.

Figure 7 Processor-shared queue



Although the class of separable networks is much larger than that of exponential server models,[4] there are still many important system and workload features that have to be sacrificed for separability. In particular, we have no results for the following conditions:

- FCFS queues with general WDDs and heterogeneous workload.
- Priorities.
- Blocking or limited access to subsystems.

- Simultaneous occupancy of resources.
- Waiting time distribution (other than mean).
- Transient solutions.

Fortunately, the modeler can often cope with those missing features by appropriate decomposition into subproblems that are compatible with the class of separable networks. Such a decomposition does not yield exact results, but in many cases it yields reasonably good approximations. We illustrate this approach in later sections.

A method for the solution of separable queuing networks

It is the object of QNET4 to embody the general class of separable queuing networks as described in the previous section. Efficiency, accuracy, and ease of use are key requirements. Although QNET4 is directed toward a methodology of computer performance analysis, it has led to a basic method of queuing network solution, rather than a special-purpose computer modeling technique. In other words, the technique being discussed does not map hardware boxes (e.g., CPUs, disks, or channels) and software structures (e.g., modules, calling sequences, or path lengths) into queuing primitives. The QNET4 research has led to a basic solution method for analytic models. This method may also be useful in conjunction with simulation methods, either to reduce the number of runs in a parametric study or to construct hybrid analytic/simulative models.

The numerical algorithms used in QNET4 are based on an exact theory.[7] At no point are such approximations as iterative methods or inaccurate decompositions used. Error analysis shows that results approximate machine precision for all practically relevant problems.

QNET4 provides a dialogue mode and a subroutine interface to user-written programs. No knowledge of the APL language is required to use our technique in the dialogue mode. There are only the following four parameter-free commands, whose functions are self-evident:

- *SETUP.*
- *EVAL.*
- *CHANGE.*
- *LIST.*

Once a command is invoked, the user is guided through a protocol of questions that are brief, but not always self-explanatory. If a user does not know how to proceed at any point in the pro-

Figure 8 Example of a protocol with error messages and the use of the *HOW* function

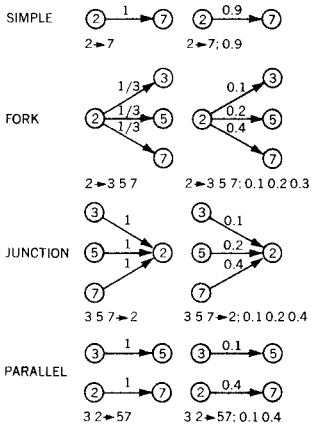
```
                EVAL
WHAT:  QUEUE  <-------------------------------------  WRONG CODE
ERR:   CODES  <-------------------------------------  ERROR MESSAGE
WHAT:  HOW    <-------------------------------------  USER ASKS FOR ASSISTANCE

SELECT CHANGE (CODE|LIST OF CODES)
QL  AVERAGE QUEUE SIZE,
TP  THROUGHPUT,                                      } TUTORIAL COMMENT
UT  UTILIZATION,
PO  AVERAGE NR OF JOBS IN GIVEN CHAIN,
RT  AVERAGE RESPONSE TIME,
ALL ALL OF THE ABOVE.

TRY AGAIN:QL  <-------------------------------------  SECOND CHANCE
```

tocol, he can call for a detailed explanation by querying with *HOW*. After the comment is printed, the user is given a second chance to answer properly.

All input data are thoroughly checked for validity. Errors are detected at the earliest possible moment. An error message is given, followed by a repetition of the faulty part of the protocol. An example of an error message together with the use of the *HOW* function is given in Figure 8.

Question answering in the dialogue mode is straightforward, except for the definition of the routing. The notation for routing transitions is slightly generalized and is the basis of the QNET4 routing description language. Figure 9 shows the four possible simple statements of the routing description language. Each statement has the following two forms: without probabilities (left column) and with explicit probabilities (right column). Simple routing description statements can be linked together as shown in Figure 10. In this fashion, it is often possible to define a whole chain in just one line. Note that the chaining of statements is optional. When the chaining of routing description statements is used, each "to part" serves as a "from part" of the next arrow.

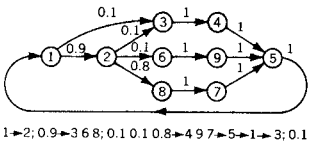
Figure 9 Routing description statements



Besides the dialogue mode, there is a set of functions designed to interface QNET4 with application programs. These functions are organized into four groups as follows:

- Initialization:
  ENV to define a QNET4 environment.
  DEFCH to define a routing chain.
  DEFQ to define a queue.
- Set or change parameters:
  CHAR arrival rate.
  CHPO population of closed chains.
  CHPR work rate (processing rate).
  CHWD work demand.
  CHRO routing.

Figure 10 Example chaining of routing description statements

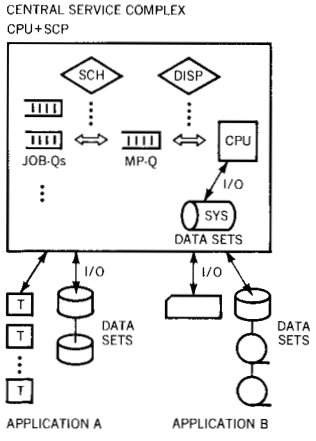


- Computation:
  SOLV to compute output tables.
- Evaluation of results:
  PO population of chains (For an open chain, *PO* returns the mean number of jobs in the chain. For a closed chain, *PO* returns the assigned number of jobs as given in *SETUP*.).
  QD marginal queue-size distribution.
  QL mean queue size.
  RT response times (For an open chain, *RT* returns the mean time a job spends in the system. For a closed chain, *RT* returns the mean time measured from the moment a job leaves a given class until the same job returns to the same class for next time.).
  TP throughput (measured in job completions/time unit).
  UT utilization.

All functions provide data and sequencing checks. For more details we refer to the user's description[8] of QNET4.

Methods of analyzing a central computer installation

Figure 11 Central computer installation



Discussed now are example models for various aspects of a central computer installation, such as the one portrayed in Figure 11. The installation consists of applications with their devices and also a central service complex. The central service complex is composed of one or several central processing units (CPUs), a system control program (SCP), and system input/output (I/O) devices. Indicated also are important queues and components of the SCP. Incoming jobs join the control complex in a job queue (JOB-Q), and are selected for execution by a component of the SCP called the scheduler (SCH). Those jobs that have been selected move to the multiprogramming queue (MP-Q), where they share the active resources, such as the CPU and the system devices. Scheduling of those resources is performed by a component termed the dispatcher (DISP). The number of jobs in the multiprogramming queue is called the level of multiprogramming. In some systems, the multiprogramming queue is divided into an IN-queue and an OUT-queue. In this case, only those jobs in the IN-queue constitute the level of multiprogramming. Movement of jobs between IN and OUT queues is called swapping.

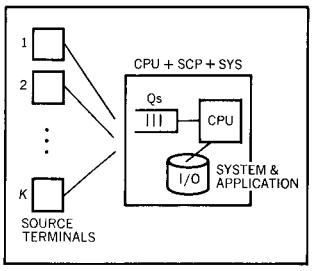
The work of the CPU is divided into system tasks and application tasks. A running application program is frequently interrupted because the CPU has to perform a system service. The nature of such an interruption is, in general, too complex to be taken into account in a high-level model. The system "overhead" is treated summarily by assigning to the processor an *effective work rate*. Such an effective work rate has to be furnished by measurement and/or empirical models. In order for

this approach to be valid, one assumes that interruptions are frequent but short, so that the processing power left over for the applications is reasonably homogeneous in time. This condition is usually well satisfied. A similar approach is used later on, when we discuss ways to model priority queues at the CPU.

finite source models

In this section, we study ways of modeling finite sources, such as a set of K terminals by an IS queue (*infinite servers*), and a closed chain. To begin, Figure 11 is shown redrawn and simplified in Figure 12 at a very gross level, where there is only one application, and attention is focused on the K source terminals that drive the system. CPU, system control program (SCP), and I/O devices are all lumped into one aggregate resource that is modeled by one queue. Assume that the system is multiprogrammed and employs an unspecified time slicing scheduling algorithm, then choose the Processor Shared (PS) queue discipline that has been shown to be an idealized model for time-sharing scheduling algorithms. On the gross level of detail of Figure 12, the whole CPU-I/O complex is characterized by one effective Millions-of-Instructions-Per-Second (MIPS) quantity.

Figure 12 Central system complex modeled as K source terminals and a central CPU-I/O complex

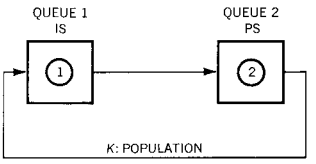


Jobs are submitted to the CPU from a group of K source terminals. During the execution of a job, its input terminal remains locked. Jobs are assumed to be statistically identical and characterized by a path length (i.e., the number of instructions to be executed). After job completion, a given terminal is ready to submit a new job. The time between job completion and new job submission is called the terminal's *think time*.

In terms of the QNET4 primitives, the model in Figure 12 has been reconfigured as shown in Figure 13. Here, the bank of source terminals is represented by the IS queue, and the number of active terminals equals the population in the closed chain. We assume the following values for the parameters:

- Number of (active) terminals: 20.
- Mean think time: 3 s.
- Effective MIPS rate: 0.5
- Mean path length: 0.1 million instructions.

Figure 13 A finite source model using QNET4



Note that the work rate and the work demand must be measured in commensurable units (i.e., if the work rate is given in MIPS, then the work demand has to be measured in millions of instructions).

The protocol of the *SETUP* command for the finite source model of Figure 13 is given in Figure 14. Results obtained by *EVAL* are shown in Figure 15. The simple finite source model with one IS queue and one other queue is also known as *machine repairman problem*, one of the first queuing models used in the analysis of

Figure 14 *SETUP* protocol for the finite source model of Figure 13

```
SETUP

CLOSED CHAINS:1
OPEN CHAINS:0
CLASSES:  2
QUEUES: 2

CHAIN: 1     (CLOSED)
ROUTING:
(1) 1→2→1
(2)

QUEUE: 1
TYPE: IS
STIMES: 3

QUEUE: 2
TYPE: PS
RATE: .5
WORK DMNDS: .1

CHAIN POP: 20
CPUT = 0.474
```

DEFINE SIZE OF NETWORK

THINK TIME (s)

MIPS RATE

PATH LENGTH (MILLIONS OF INSTRUCTIONS)

NUMBER OF TERMINALS

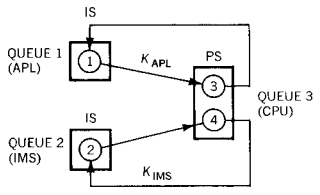
Figure 15 List of results (*EVAL*) for the finite source model of Figure 13

```
          EVAL
WHAT:     ALL

PO:         20                CHAIN POPULATION

Q1    QL:       14.3
Q2    QL:       5.68          AVERAGE QUEUE SIZE

Q1    UT:       0.716
Q2    UT:       0.954         UTILIZATION (FRACTION OF BUSY TIME)

Q1    TP:       4.77
Q2    TP:       4.77          THROUGHPUT (JOBS/s)

CLASS:          1 2
RT:         1.19 3            RESPONSE TIME(s) (CLASS 1 AND CLASS 2)
```

computer systems.[9] The last line in Figure 14, which is the amount of CPU time used, is also a message to acknowledge that the system is now ready to list results.

It is important to note that both the distribution of think times and the distribution of path length can be general. Only in the case of FCFS queuing at the CPU is one forced to make an exponential assumption.

Figure 16 Finite source model with two chains



The next example attempts to bring greater realism to the finite source system model by considering two subsystems, e.g., IMS and APL. Such systems can be represented by two closed chains, as shown in Figure 16. Assume that both chains are of the same priority level at the CPU and assume the following parameter values:

- Number of APL terminals: 15.
- Number of IMS terminals: 5.
- Mean think time APL: 1 s.
- Mean think time IMS: 15 s.

Figure 17 *SETUP* protocol for finite source model with two chains

```
      SETUP

CLOSED CHAINS:2
OPEN CHAINS:0
CLASSES:  4
QUEUES: 3

CHAIN: 1     (CLOSED)
CLASSES: 1 3
ROUTING:
(1) 1→3→1
(2)

CHAIN: 2     (CLOSED)
CLASSES: 2 4
ROUTING:
(1) 2→4→2
(2)

QUEUE: 1
TYPE: IS
CLASSES: 1
STIMES: 1

QUEUE: 2
TYPE: IS
CLASSES: 2
STIMES: 15

QUEUE: 3
TYPE: PS
RATE: .4
CLASSES: 3 4
WORK DMNDS: .01 .2

CHAIN POP: 15 5
CPUT = 0.763
```

SIZE OF NETWORK

ROUTING, APL SYSTEM

ROUTING, IMS SYSTEM

APL TERMINALS

THINK TIME APL (s)

IMS TERMINALS

THINK TIME IMS (s)

CPU

EFFECTIVE MIPS RATE

PATH LENGTH (MILLIONS OF INSTRUCTIONS)

NUMBER OF TERMINALS

CPU TIME FOR RUN (s)

- Mean path length APL: 0.01 millions of instructions.
- Mean path length IMS: 0.2 millions of instructions.
- Effective MIPS rate: 0.4.

The *SETUP* protocol for the two-chain finite source model is given in Figure 17, and the results obtained by *EVAL* are shown in Figure 18. Note that in the output, queue length, throughput, and response times are broken down into classes.

two-chain finite source model

In the two-chain model, we have made use of the class and chain concepts in order to model heterogeneous workloads and disjoint subsystems. The absence, however, of priority queue disciplines from the class of separable networks has forced the assumption that both APL and IMS are on the same priority level. In many instances, such an assumption is not realistic. For example, assume that the APL system takes preemptive priority over the IMS system. Thus, the APL system "sees" the full processing rate C of the CPU, a premise that permits the construction of an individual finite source model similar to the basic finite source model in Figure 13. The IMS system receives the remaining work capacity that on the average is given by the following equation:

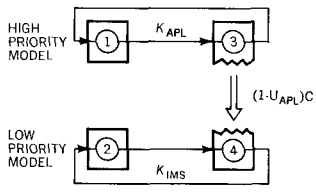
$$C_{\text{IMS}} = (I - U_{\text{APL}})\ C \qquad (4)$$

where U_{APL} is the utilization obtained from the APL model. The

Figure 18 List of results (*EVAL*) for finite source model with two chains

```
          EVAL
WHAT:     ALL

CH1       PO:        15
CH2       PO:         5                    <-- AVERAGE CHAIN POPULATION

Q1        QL:      14.3
Q2        QL:      4.71
Q3        CLASS:      3       4  TOT       <-- AVERAGE QUEUE SIZE
          QL:      0.677  0.293  0.97

Q1        UT:      0.955
Q2        UT:      0.941                   <-- UTILIZATION (FRACTION OF BUSY TIME)
Q3        UT:      0.515

Q1        TP:      14.3
Q2        TP:      0.314
Q3        CLASS:      3       4  TOT       <-- THROUGHPUT (JOB/s)
          TP:      14.3  0.314  14.6

CH1       CLASS:          1  3            <-- RESPONSE TIMES, CHAIN 1 (s)
          RT:      0.0473  1

CH2       CLASS:          2  4            <-- RESPONSE TIMES, CHAIN 2 (s)
          RT:      0.934  15
```

Figure 19 Decomposition of a finite source model with two chains into a high and low priority model



processing of IMS transactions, however, is not homogeneous; bursts of processing alternate with idle periods. If we ignore the effects of this nonhomogeneity, we can construct a second finite source model for IMS with a processing rate given by Equation 4. This decomposition of the original problem into parts that are compatible with the QNET4 class of networks is shown schematically in Figure 19.

Clearly the assumption of smoothed processing in the low-priority model introduces an error that can be estimated by replacing the finite source with an infinite (or Poisson) source and using results of the Poisson arrival process, exponential service process, one server, with priority discipline (M/M/1/PRI) queue theory. There results the following expression for the relative error of the mean waiting time:

$$E = \frac{(m_3/m_4)\rho_3}{1-\rho_3+(m_3/m_4)\rho_3} \tag{5}$$

where m_3 is the mean path length APL, m_4 is the mean path length IMS, and

$$\rho_3 = \frac{(\text{throughput APL})\,m_3}{C} \tag{6}$$

Equation 5 reveals that E is always positive, which means that the smoothing out of the IMS processing produces optimistic results for the response times. The error, however, never exceeds one-hundred percent and remains small if ρ_3 is small (light APL load), and/or if the APL path length is much shorter than the IMS path length.

In most practical cases, high-priority transactions tend to be shorter than low-priority transactions, and, hence $m_3/m_4 < 1$. In

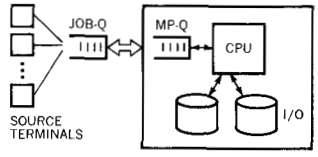
Table 1 Results of the hierarchical research QNET4 model compared with simulation results obtained by APLOMB

		$C = 0.4$	$C = 0.2$
Mean queue Size APL	QNET4	.537	1.6
	APLOMB	.526, .539, .553	1.58, 1.62, 1.65
Mean queue Size IMS	QNET4	.313	1.44
	APLOMB	.262, .306, .349	1.42, 1.55, 1.68
Error (Equation 5)		2.75%	9.4%

the given example, we find $\rho_3 = 0.363$, $m_3/m_4 = 0.05$, and E = 2.75 percent by Equation 5.

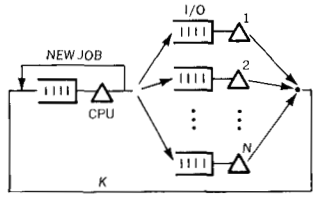
We have compared the analytical modeling (QNET4) approximate solution with simulation results obtained by a queuing network simulation method that is known as APLOMB.[10] Results summarized in Table 1 show that the simple hierarchical QNET4 model is as accurate as simulating 100,000 transactions. Execution times of APLOMB simulations compared with QNET4 are in the approximate ratio of fifty to one. The three numbers in the APLOMB row represent a 95 percent confidence interval with a point estimate.

Figure 20 Computer system as seen by the manager of a multiprogramming queue



Multiprogramming was introduced to increase resource utilization by overlapping CPU processing with I/O operations. A simple cyclic queue model to evaluate multiprogramming performance is among the earliest analytic computer models.[11] This cyclic queue model has been generalized into the so-called central server model.[12]

Figure 21 A central server model



We now concentrate on the subsystem CPU, I/O, and dispatcher as shown in Figure 20, where MP-Q is a multiprogramming queue. The level of multiprogramming must be carefully controlled in order to optimize performance, and is typically much smaller than the number of source terminals.

central server model

In the central server model, interest does not focus on source terminals and job queue (JOB-Q) scheduling. In fact, a fixed level of multiprogramming K is assumed. It is also useful to break up the multiprogramming queue into a CPU queue and several device queues, as illustrated for a typical central server model with N I/O devices in Figure 21.

The assumption of a fixed multiprogramming level leads to a closed network whose population equals the multiprogramming level. In Figure 21, each device is modeled by a single queue, a representation that requires the assumption that channels are not heavily utilized (otherwise queues representing channels would have to be introduced). The cyclic routing between CPU and I/O devices originates in the fact that computation intervals (also called CPU bursts) and I/O accesses alternate for a given job. It is assumed that there is no CPU-I/O overlapping for individual jobs, an assumption that has been shown to be empirically valid in many cases. The routing transition around the CPU labeled "new job" represents the event that a job has finished processing and departs. To maintain the level of multiprogramming, a new job must be immediately injected. Since jobs are statistically identical, we can achieve instantaneous replacement by the routing loop around the CPU. The throughput in that loop is the job completion rate of the system.

Workload and system parameters of the central server model are as follows:

- Level of multiprogramming K.
- Mean path length of a job is m millions of instructions.
- Mean path length between I/O operations to device n is m_n millions of instructions.
- Access time of device n is t_n s.
- Effective MIPS rate of CPU is C.

Under exponential assumptions, we can derive the path length of a CPU burst as follows:

$$m_{\text{cpu}} = \frac{1}{1/m + 1/m_1 + \cdots + 1/m_N} \tag{7}$$

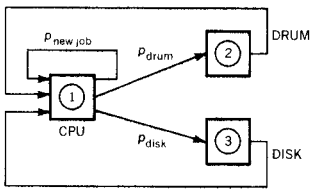
The probability that after completion of a CPU burst a job branches to I/O device n $(n = 1, 2, \cdots, N)$ is

$$p_n = m_{\text{cpu}}/m_n \tag{8}$$

The probability of the branch "new job" is expressed as follows:

$$p_{\text{new job}} = m_{\text{cpu}}/m \tag{9}$$

Figure 22 A central server model expressed in QNET4 methodology



An APL program for the simple central server model of Figure 22 is given in Figure 23; use of the QNET4 subroutine interface is made.

There exists a large literature on the central server model, yet validation results are relatively rare. There is increasing evidence that the central server model provides good results, even in cases where the distributions deviate from the exponential assumptions.[13-15] This may be explained by the fact that practical

Figure 23 A. APL function for the central server model of Figure 22, and B. results for a multiprogramming level of 10

```
     ∇ Z←CENTRALSERVER K
[1]  ⍝ INPUT PARAMETERS
[2]  ⍝   K:  LEVEL OF MULTIPROGRAMMING
[3]  ⍝   MJOB: JOB PATH LENGTH
[4]  ⍝   MDRUM: PATH LENGTH BETWEEN I/O TO DRUM
[5]  ⍝   MDISK: PATH LENGTH BETWEEN I/O TO DISK
[6]  ⍝   C: EFFICTIVE MIPS RATE CPU
[7]  ⍝   TDRUM:  MEAN DRUM ACCESS TIME
[8]  ⍝   TDISK:  MEAN DISK ACCESS TIME
[9]  ⍝ RESULTS:  Z: THROUGHPUT
[10] ⍝
[11] ⍝ PATH LENGTH OF CPU BURST
[12]  MCPU←1÷((1÷MJOB)+(1÷MDRUM)+(1÷MDISK))
[13] ⍝ PROBABILITY FOR BRANCH <<NEW JOB>>
[14]  PNEWJOB←MCPU÷MJOB
[15] ⍝ PROBABILITY FOR BRANCH CPU→DRUM
[16]  PDURM←MCPU÷MDRUM
[17] ⍝ PROBABILITY FOR BRANCH CPU→DISK
[18]  PDISK←MCPU÷MDISK
[19] ⍝ SET NUMBER OF QUEUES(N) AND
[20] ⍝      NUMBER OF CLASSES (M)
[21]  M←N←3
[22] ⍝ SET NUMBER OF CLOSED(LC) AND
[23] ⍝      OPEN CHAINS (LO)
[24]  LC←1
[25]  LO←0
[26] ⍝ DEFINE ENVIRONMENT
[27]  ENV
[28] ⍝ SET ROUTING
[29]  1 CHRO 3 3 ⍴PNEWJOB,PDURM,PDISK,1,0,0,1,0,0
[30] ⍝ SET PROCESSING RATE QUEUE 1 (CPU)
[31]  1 CHPR C
[32] ⍝ SET WORKDEMAND QUEUE 1
[33]  1 CHWL MCPU
[34] ⍝ SET SERVICE TIMES OF DEVICES
[35]  2 CHWL TDRUM
[36]  3 CHWL TDISK
[37] ⍝ SET LEVEL OF MULTIPROGRAMMING
[38]  CHPO K
[39] ⍝ COMPUTE INTERNAL TABLES
[40]  SOLV
[41] ⍝ COMPUTE THROUGHPUT OF BRANCH <<NEW JOB>>
[42]  Z←PNEWJOB×TP 1
     ∇
```

(A)

```
MJOB←.1
MDRUM←.005      PATH LENGTHS (MILLIONS OF INSTRUCTIONS)
MDISK←.01
C←.75           EFFECTIVE MIPS RATE
TDRUM←.03       DEVICE ACCESS TIMES (s)
TDISK←.1
CENTRALSERVER 10

0.997    JOB COMPLETION RATE (JOBS/s)
```

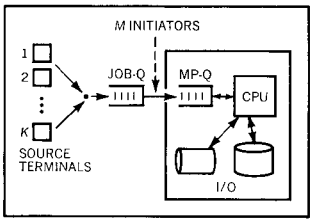
(B)

scheduling disciplines tend to be more robust than the FCFS discipline. Additionally, deviations from the exponential distribution at the CPU (coefficient of variation greater than one) and at the I/O devices (coefficient of variation less than one) have an opposite effect on performance measures and tend to cancel each other. The central server example considered here has been kept simple. Clearly, chains and classes can be used to model a heterogeneous workload, and decomposition may be used in the case of priorities.

limited access subsystems and blocking

The finite source model and the central server model both represent certain aspects of the basic computer installation shown in Figure 11. We shall now combine both models to arrive at a more realistic representation of the overall system. For simplicity, assume that the workload consists of statistically identical jobs that are generated by a set of K terminals. Jobs join the sys-

Figure 24 Finite source central server model with K terminals and M initiators



tem in one job queue. There are M initiators that select jobs from the job queue for inclusion in the multiprogramming set. Clearly the number of initiators puts an upper limit on the multiprogramming level. Figure 24 represents the hypothetical finite-source-central-server computer system.

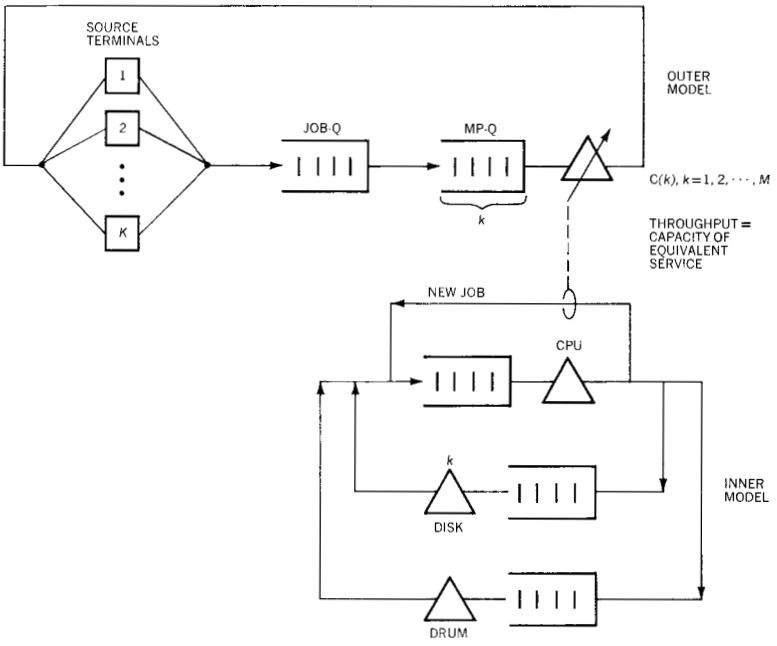
Typically, the number of active sources is much greater than the number of initiators (i.e., $K > M$). An attempt to map the system structure of Figure 24 into a QNET4 model immediately raises the problem of limiting the level of multiprogramming to at most M.

Use of a decomposition technique is again recommended. Observe that time constants of terminals and the scheduler are typically one or two orders of magnitude larger than those of the CPU and I/O devices. Therefore, the CPU-I/O subsystem reaches equilibrium much faster than subsystems that consist of terminals and the job queue. Therefore, try to represent the CPU-I/O complex by one equivalent queue in the terminal-scheduler submodel. The rate of this equivalent queue is obtained from a central server model. The throughput of the central server model determines the rate of the equivalent queue. Since throughput depends on the level of multiprogramming, we have a queue-dependent server in the finite source model that represents terminals and schedulers. This hierarchical decomposition is shown schematically in Figure 25. We call the finite source model the *outer model* and the central server model the *inner model.* The inner model of Figure 25 is the same as the one of Figure 22.

The hierarchical decomposition of Figure 25 is also a possible device to combine QNET4 with simulation. Suppose there is a complicated algorithm for the computation of priorities in the job queue. If the goal is to evaluate such an algorithm, one cannot use QNET4 for the outer model. Given that the central server model is adequate, one can reduce the complexity of a simulation model and its necessary running time by representing the CPU-I/O complex by an equivalent queue exactly as in Figure 25. Savings are particularly great if the time constants of outer and inner models differ markedly (which is the necessary condition for the hierarchical decomposition to work).

An error analysis of the hierarchical model is difficult. Under the additional assumptions of an infinite source and a single I/O device, an analytical solution has been obtained.[16] This solution confirms the intuitively appealing requirement that the inner model be faster than the outer model. For a given and fixed load and CPU and I/O device, we find that the exact solution converges to the one of the hierarchical QNET4 models if the average number of I/O accesses tends to infinity. (As a consequence, the speeds of the CPU and I/O device also tend to infinity.)

Figure 25 Hierarchical central server finite source model



Concluding remarks

Considerable progress has been made in the use of queuing networks for high-level system models. We have discussed an example of a computer system model that might be useful in a configuration study or as a host model in teleprocessing network design. Other areas where queuing network models have been successful are the following:

- Storage hierarchy design.
- System bus design.
- Multiprocessor system evaluation (including storage interference and locking).
- Teleprocessing network design.

Present day state of the art in solutions of queuing systems still does not provide for some important system features, such as priority disciplines and blocking. As far as general service-time distributions (or work demand distributions) are concerned, we have witnessed some considerable progress in recent years. The inclusion of the Processor Shared servers (PS), Infinite Servers (IS), and Last Come, First Served (LCFSPR) queue disciplines into the class of solvable models, and the resulting understanding of the phenomenon of robustness has largely reduced the suspicion with which the exponential distribution was formerly treated.

Approximate decomposition and smoothing out of discontinuous service have proved to be powerful techniques for overcoming some of the shortcomings of the basic model. The processor sharing approximation to the round-robin scheduling discipline and the treatment of preemptive priorities in the multichain central server model have been examples of smoothed processing. Decomposition has been used successfully for modeling various types of systems, such as the following:

- Blocking limited access to subsystems.
- Simultaneous occupancy of resources (e.g., channels that are held simultaneously with the devices).
- Multiprocessor systems with locking and storage interference.

Decomposition also allows combinations of analytic and simulation models. References 17 and 18 are additional discussions of modeling and queuing systems on an introductory level.

ACKNOWLEDGMENT

The author is in debt to the many people in IBM who have helped in the research and debugging of QNET4. In particular, the contributions of D. Wong, H. Kobayashi, K. F. Finckemeier, A. Mecklenburg, S. Kiss, G. S. Shedler, and R. Downs are deeply appreciated.

CITED REFERENCES

1. L. Kleinrock, *Queuing Systems, Volume 1: Theory*, John Wiley and Sons Inc., New York, New York (1975).
2. D. R. Cox, "A use of complex probabilities in the theory of stochastic processes," *Proceedings of the Cambridge Philosophical Society* **51,** 313–319 (1955).
3. M. Reiser and H. Kobayashi, "Accuracy of the diffusion approximation for some queuing systems," *IBM Journal of Research and Development* **18,** No. 2, 110–124 (1974).
4. J. R. Jackson, "Jobshop-like queuing systems," *Management Science* **10,** 131–142 (1963).
5. F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed and mixed networks of queues with different classes of customers," *Journal of the ACM* **22,** No. 2, 248–260 (1975).
6. M. Reiser and H. Kobayashi, "Queuing networks with multiple closed chains: Theory and computational algorithms," *IBM Journal of Research and Development* **19,** No. 3, 283–294 (1975).
7. M. Reiser, *Numerical Methods in Separable Queuing Networks*, Report RC5842, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598
8. M. Reiser, *QNET4 User's Guide*, Report RA71, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York, 10598.
9. A. L. Scherr, *An Analysis of Time-Shared Computing Systems*, MIT Press, Cambridge, Massachusetts (1967).
10. C. H. Sauer, "Simulation analysis of generalized queuing networks," *1975 Summer Simulation Conference*, 75–81 (1975).
11. D. P. Gaver, "Probability models for multiprogramming computer systems," *Journal of the ACM* **14,** No. 3, 423–438 (1967).

12. J. Buzen, *Queuing Network Models of Multiprogramming*, Ph.D. Thesis, Division of Engineering and Applied Science, Harvard University, Cambridge, Massachusetts (1971).
13. W. Chiu, D. Dumont, and R. Wood, "Performance analysis of a multiprogrammed computer system," *IBM Journal of Research and Development* **19,** No. 3, 263–271 (1975).
14. T. Giampo, "Validation of a computer performance model of the exponential queuing network family," *Proceedings of the International Symposium on Computer Performance, Modeling, Measurement and Evaluation*, 44–58, Harvard University, Cambridge, Massachusetts (1976).
15. C. A. Rose, "Validation of a queuing model with classes of customers," *Proceedings of the International Symposium on Computer Performance, Modeling, Measurement, and Evaluation*, Harvard University, Cambridge, Massachusetts (1976).
16. M. Reiser and A. G. Konheim, *Finite Capacity Queuing Systems with Applications in Computer Modeling*, Report RC5827, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.

Reprint Order No. G321-5039.