Various aids and tools are used in capacity planning. One such aid, an analytic model, is discussed in this paper. Both the decisions made in the development of an aid and the way the aid is used are examined. Characteristics of a good planning aid are emphasized with the analytic model serving as the example.

Modeling considerations for predicting performance of CICS/VS systems

by P. H. Seaman

This paper discusses an analytic model[1] currently used in the design of and capacity planning for on-line CICS/VS (Customer Information Control System/Virtual Storage)[2] systems. The discussion focuses on the characteristics that typify a good planning aid in general, employing aspects of the CICS model as specific examples. These characteristics are examined from two perspectives: first, an internal view of specific design decisions that must be made by an aid developer, followed by an external view of the way the aid should be used by the planner.

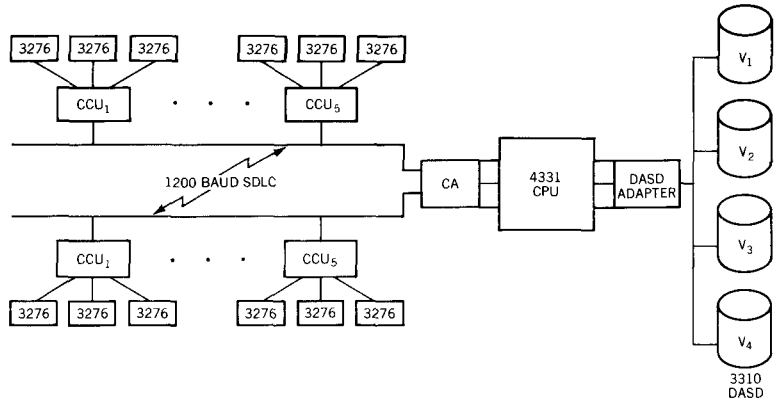
Description of the model

To facilitate the reader's understanding of the model, the input it requires, and the operations it performs, a simple inquiry system for a credit application will be described. The system is shown schematically in Figure 1.

This system includes a central processing unit (CPU) with four direct access storage devices (DASD) attached. Inquiry messages enter the system via remote terminals attached to two communication lines by means of cluster control units (CCUs). The line traffic enters the CPU through a communications adapter. The CPU software consists of CICS/VS under the Disk Operating System (DOS), including standard IBM access method packages for line control and DASD control. The principal application data set is the customer credit file spread over the four DASD units.

The particular aid that models such a CICS system is implemented as an interactive APL (A Programming Language) program. The

Figure 1 Simple inquiry system



program estimates the performance of a specified CICS system, calculating transaction response times and utilizations for various system resources. Input to the program consists of (1) a description of the system configuration and various data sets to be referenced, (2) representations of the application programs to process arriving transactions, and (3) the arrival rates of those transactions to the system from both local and remote terminal locations.

The configuration shown in Figure 1 can be defined by a few statements that contain various CPU specifications, device type designations, file names, and key parameters, such as access method, logical record length, and number of records, along with the layout of the data set extents on particular storage units.

Transactions are then defined, the definitions including transaction name, type, number of bytes in the initial input message, etc. With each transaction, a sequence of "macro" operations is specified, outlining the various CICS activities that the transaction will invoke during its sojourn in the central site. To a great extent, these "macros" resemble actual CICS macros (macroinstructions). Also included are special macros, representing general system activity, such as PROCESS N, representing the time to execute N machine instructions, or SIO A (Start I/O A), which generates a non-CICS I/O access to file A. These macro sequences resemble the application programs by which the transactions are processed, although the logical flow is not adhered to as it would be in a particle-flow simulator, and there is not a one-to-one relationship since a real transaction may be processed by several application programs. A typical sequence of macros for one transaction might be

```
BMSI                    Basic mapping input
G           A           Get a record from file A
```

PROC	10 000	Execute 10 000 instructions
TSP	AUX, 50	Put 50 characters on auxiliary temporary storage
BMSO		Basic mapping output
TCW	200	Write 200-character message to terminal
END		End of sequence

Finally, traffic rates are specified, indicating on which communication lines and local terminals the various transactions originate. Each transaction type can originate from many different locations. In addition, the specified base rates can be augmented by a set of traffic multipliers to assist in traffic growth studies.

After all this data has been entered, the transaction macro sequences are scanned by the program, and, together with the associated transaction rates, summaries are accumulated relating to specific system activities. Three essentially independent queuing formulations make up the bulk of the internal calculations—a communication line model, a DASD model, and a CPU model. The line model takes the summarized message rates and sizes and calculates line utilizations, waiting delays, and transmission times, based on the line configuration specified. The DASD model takes the summarized access rates and sizes relating to the specified CICS files and system data sets and calculates device and channel utilizations as well as data access times. The CPU model, using an internal table of path lengths for all the various CICS activity elements, develops a total path length for each transaction type, and from this path length and the summarized transaction rates, calculates the CPU utilization and waiting times to gain control of the processor.

Finally, the macro sequences are rescanned and the individual pieces, including the waiting times generated by the three queuing models, are added together to produce total average response times for each specified transaction type. All these performance statistics are then gathered together and reported to the user in a concise format, including an analysis of apparent problem areas, if any.

Such is the nature of the CICS planning aid. Some of the considerations that went into the development of the tool itself are now discussed.

Model simplicity versus useful accuracy

In a capacity planning environment, where there is a large amount of uncertainty in predicting future requirements, a system model of the foregoing type entails many trade-offs. The model

should be reasonably accurate, but it need not track the real system with the degree of fidelity required, say, of a detailed simulator used for integration testing or system tuning. The model developer must balance the need for accuracy against the possibilities of obtaining the input data required to produce that accuracy. Ease of use and maintainability considerations call for the simplest models consistent with planning requirements. The key to a good planning model is first understanding its end use and the accuracy required, and then integrating the divergent pieces and levels of detail into a consistent whole to meet this use. Examples of how three system features came to be represented as they now exist in the CICS model may illustrate this point. The three features to be exemplified are data set placement, queuing of tasks for system resources, and a batch workload.

Example 1

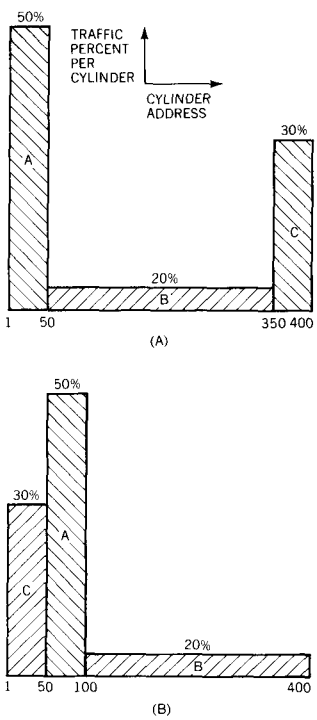
First consider data set placement. It is well known that cylinder location of data sets on DASD affects seek time and, therefore, data access time. In an effort to improve access time, a system is often tuned by attempting to group together the data sets with the highest frequency, as shown in Figure 2.

The average seek time to a particular data set i may be estimated in the following way:

$$\text{Seek } i = \sum_{j=1}^{n} p_{ij} \cdot S_{ij} \qquad 1 \leq i \leq n \tag{1}$$

where n is the number of data sets on the storage device; p_{ij} is the probability of moving from data set j to data set i; and S_{ij} is the average time to move between those same two data sets. The value of p_{ij} may normally be computed as $p_i \cdot p_j$, where p_i and p_j are the probabilities of accessing data sets i and j. For current storage devices, a good estimate for S_{ij}, when $j \neq i$, is the arm motion time between the midpoints of the data sets i and j. This can be determined from the seek characteristic curve for the device. For average motion time within data set i, that is, S_{ii} when $j = i$, a good estimate is the arm motion time to move one-third of the width of data set i. This is referred to as the "one-third rule" and is often used as a rule of thumb for the whole device when no detailed information is available.

Figure 2 Two arrangements of three data sets on an IBM 3330 disk pack: (A) Arrangement 1 (initial), (B) Arrangement 2 (improved)



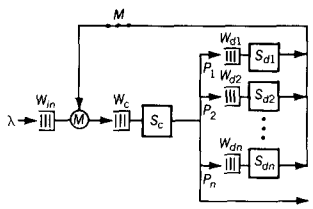
Based on this analysis, the average seek times to the three data sets A, B, and C, arranged as shown in Figure 2A, are 26, 30, and 32 milliseconds, respectively, with an overall average of 29 milliseconds. In the rearrangement shown in Figure 2B, the average seek times to the same data sets are 19, 31, and 20 milliseconds, with the overall average being 22 milliseconds. If the specific frequency pattern is ignored altogether and a uniform distribution is assumed over all 400 occupied cylinders, the average seek time to any data set would be estimated from the "one-third rule" as 28 milliseconds.

The question facing the modeler is whether to ask the user for cylinder address information in order to employ this detailed model. Also, the extra logic that is required to calculate and manipulate the multiple seek values must be assessed in terms of maintainability and execution time. In the case of the CICS model, this detail was not included. (Actually, a compromise was struck whereby the uniform assumption includes only the number of active cylinders defined by the user.) While the data layout is available for installed systems, it is usually mere guesswork for new applications. In addition, the majority of CICS systems are either line bound or CPU bound so that increased accuracy in the I/O area does not add much information from a capacity planning viewpoint. Further, the uniform assumption is generally on the conservative side so that in cases where I/O presents a bottleneck, the model will report saturation effects at lower traffic rates than would be experienced by the real system. It is important to remember that the model is used principally for planning purposes, such as comparing the IBM 3310 with the IBM 3330 storage device, and not for tuning a system. One can validly compare the performance of two disk units without detailed placement data if the same uniform assumption is made in both cases. This relieves the user from burdensome input detail, which may be difficult to determine accurately, as well as making the program logic simpler and faster.

Example 2

Another feature exemplifying model simplicity is the finite source queuing model used in the program. A schematic of the CPU with its auxiliary storage is illustrated in Figure 3.

Figure 3 Schematic of transaction processing flow in CICS



Tasks enter the system with rate λ. (Although there are a finite number of input terminals in the system, the total number is usually large (>100) compared to the active number (<10) with tasks being processed in the system, so the arrival rate is assumed to come from an infinite population.) Up to M concurrent tasks will be accepted; any additional tasks must wait in an input queue until one or more of the active tasks are complete. Each of the active tasks is processed by cycling about the main loop, alternately requesting CPU service and a data access from one of the storage devices available. The frequency of access to each storage device, given as P_1 through P_n, is a function of the transaction mix and data set assignments. Each request for service or data may incur a wait for prior requests. The purpose of the model is to estimate these waiting times so that their contribution to total transaction response time can be ascertained.

This example belongs to a class of queuing models of recent interest called "central server" models.[3-5] In most central server models that have been considered, the external traffic λ is not present, and the number of internal concurrent tasks is a fixed number M; that is, they are essentially representing batch systems. The exact

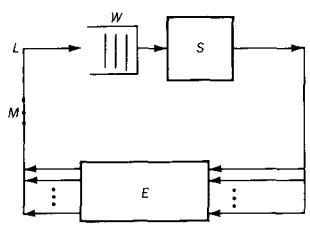
solution usually has too many restrictive assumptions or else is very complex, making it time-consuming to run and difficult to keep current, trying to include all the little peculiarities that always arise when modeling specific systems.

Three approaches to an approximate solution were considered. In the first and simplest,[6] the cyclic nature of the processing loop was ignored and all service nodes were treated independently. From the external transaction rate λ driving the system, the arrival rate for each node can easily be derived by multiplying λ times the number of visits made to the node by the average transaction during its processing. Then the waiting time in front of each node can be calculated assuming the node to be a single-server queue fed by an infinite source.

The node utilizations are directly ascertained, while the effect of priorities can easily be added to the model. However, W_{in}, the initial wait in the input queue, disappears, that delay being absorbed into the internal queues. Thus, the effect of different levels of task concurrency, M, cannot be appraised. This simplistic model serves as an upper limit to the real system, representing the case when the maximum M becomes very large.

A second approach assumed the node throughput rates were known as before, but the *average* value of concurrency $\bar{M}$ was the important factor to be determined. The value of $\bar{M}$ was approximated by assuming each node to be a closed queue known as a "machine repairman" model, shown in Figure 4.

Figure 4 "Machine repairman" model



Classically in the "machine repairman" model, a fixed number of M machines periodically request service S from a single repairman. If the repairman is busy, the machines must wait a time W. Upon completion of service, the machines operate concurrently for a period E until they again require service. If appropriate distributions for S and E are assumed, the calculation of the average waiting time experienced by a machine requesting service may be expressed as a function[7]

$$W = f(S, E, M) \tag{2}$$

With the waiting time W thus calculated, the throughput of the repair facility (e.g., number of machines repaired per hour) can be expressed as

$$L = M/(W + S + E) \tag{3}$$

This formulation may be applied to the "central server" model by considering each of the $n + 1$ nodes in turn as a "task processor" equivalent to a "repair facility," with a processing time S_i $(1 \leq i \leq n + 1)$ and a fixed throughput L_i determined by the external driving rate λ. Each node will be recurrently visited by the $\bar{M}$ tasks after spending time E_i elsewhere in the system. In the

special case of the CPU node, elsewhere time E_c can be calculated as the average disk access time. The successive iteration of Equations 2 and 3 for all nodes and the employment of the resulting value for E_c allows values for waiting times at all nodes as well as the average number of current tasks $\bar{M}$ to be approximated. By placing a restriction that $\bar{M} \leq M$, the effect of restricting task concurrency can be shown. Finally, using a multiserver queue with M servers, a value for W_{in} can be calculated.

However, this model has several deficiencies. The iterations require considerable execution time, with no commensurate gain in accuracy. In fact, in many cases the input waiting time is an order of magnitude too small because of the assumption that the number of concurrent tasks may be estimated by the average value. Actually, the dynamically varying number of tasks has a much greater effect on the system than what is estimated by using the average number of tasks.[8]

The third approach to approximating the "central server" model consisted of a slight modification to the second approach. Whereas the number of concurrent tasks was considered unknown in the preceding method and the average value determined iteratively, the value M in this third approach was assumed to be the maximum number of active tasks allowed, AMXT in CICS terminology. The M tasks were assumed to be somewhere in the system, though perhaps not currently active in the main processor or associated disk storage. From the viewpoint of each individual node, M was the maximum queue size that could ever be experienced. Further, the calculated throughput for all nodes in the main loop no longer was balanced against the specified throughput. It was this latter balancing operation that required the outer iteration in the second approach.

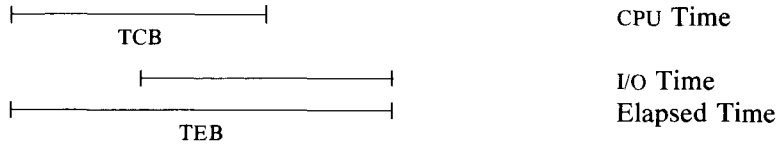
As before, every node was considered independently as a "machine repairman," operating according to Equations 2 and 3. However, now M, as well as S_i and L_i, were given so that values for W could be determined for all nodes in one pass, without further iteration. The initial input wait W_{in} was again calculated for a multiserver queue with M servers. Since the overall task-processing time was guaranteed a higher estimate, the input wait was greater as required.

This last approach was adopted for the model because (a) it adequately represented the effect of restricting task concurrency, (b) it was simple to implement and maintain, and (c) it was easily extended to include other special effects, such as task priorities and partition lockout due to paging.

Example 3 A final example illustrating the choice of model detail consistent with the end use of capacity planning is the representation of a batch workload within the CICS model.

It was realized that the low-priority batch jobs had little effect on the performance of the higher-priority CICS transactions, except through activity by the batch workload on shared channels and storage units. However, the effect of the CICS workload on batch throughput was substantial. Little information concerning the batch work was needed to adequately represent its effect on CICS, but to model the reverse interaction accurately required the definition of batch jobs in great detail. Fortunately, this accuracy was not necessary. What was needed was to develop a representation requiring the absolute minimum of input that would give a reasonable sense of direction concerning the amount of batch work that might be handled by a processor over and above a given CICS workload.

Given this premise, the batch workload was viewed in the most summary form possible and characterized as follows:

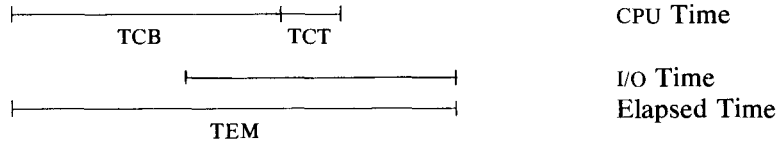


The CPU utilization UBO for this batch workload without on-line interference is

$$UBO = TCB/TEB \tag{4}$$

The unoverlapped I/O time is given by $(1 - UBO) \times TEB$.

Adding a high-priority CICS partition with CPU time TCT then yields the following representation for the mixed workload:



The new elapsed time, TEM, for a typical batch job is longer than the former value, TEB, because of delays caused by the on-line load. The CPU utilization due to the batch portion is

$$UB = TCB/TEM \tag{5}$$

The CPU utilization due to CICS is

$$UT = TCT/TEM \tag{6}$$

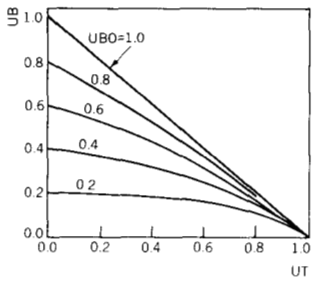
The unoverlapped batch I/O time for the mixed workload is assumed to decrease by the factor $(1 - UT)$. That is, as the on-line load increases, the unoverlapped I/O time shrinks to zero. With this assumption, the total elapsed time for a mixed run is

$$TEM = TCB + TCT + (1 - UBO) \cdot TEB \cdot (1 - UT) \tag{7}$$

Combining the results of Equations 4, 5, 6, and 7 produces CPU utilization for a batch workload in a mixed environment:

$$UB = \frac{UBO \times (1 - UT)}{1 - UT \times (1 - UBO)} \tag{8}$$

Figure 5 Batch utilization UB versus on-line utilization UT for several values of batch-only utilization UBO



UB is smaller than the batch-only utilization UBO because the batch workload is extended by the on-line work, slowing down the rate at which processing and data resources are accessed. Equation 8 is plotted in Figure 5 for several values of UBO. Batch throughput, or the number of jobs per hour that are processed by the system when CICS is present, is directly related to the batch-only system by the ratio UB/UBO.

Even with such minimal input requirements, one apparent difficulty with the batch model is in obtaining a value for UBO from a mixed system already installed for which a study regarding new applications is desired. It would probably be impossible to stop the CICS system temporarily while batch measurements were taken. However, the current measurements for CPU utilization due to CICS (UT) and the batch workload (UB) can be entered in Equation 8 and the value for batch-only utilization (UBO) derived. Then, with the use of this value, a new level of UB can be estimated from the model as UT changes due to new CICS applications being specified.

Another potential weakness with this model is that it works best when data and system files for the on-line and batch applications are kept on separate storage units. However, this stricture is usually met in real cases since it is highly desirable for reasons of security and control as well as performance.

Several measurements from actual systems have since shown the model to be reasonable in predicting batch slowdown. When the small amount of input required is considered, this has made the batch model very useful for planning purposes.

Now, having viewed this performance predictor from the standpoint of a developer, we shift our focus to view the program as an aid in the planning process.

Use of the model

With such a model of CICS available, one must consider how best to use it as a planning tool. The usual starting point is the representation of an installed system. Most customers requiring CICS planning have an installed system that is either to be upgraded or to have a new application installed or both. By modeling the current system, the performance estimates can be verified from available measurements. This gives both a marketing representa-

tive and his customer confidence in what the model is telling them. It further forms a solid base on which to build any proposed extensions.

A very important point to be made here, which applies to the user in the same way it did to the model developer, is *Think Simple*! A wealth of detail is only a welter of facts if one does not understand how the details relate. For example, the real system may have 89 transaction types, but one should not try to model them all. It is not necessarily true that by representing all 89, the model would be more realistic. Many compromises would have to be made in representing each case, whereas the overall model itself is a patchwork of compromise. Therefore, it would be very difficult to comprehend the final result. It is important to remember that in many large systems less than 20 percent of the causes usually account for more than 80 percent of the effects. Determine the dozen or so really significant transactions, based either on occurrence or processing activity, and model them. Then add one or two miscellaneous transactions to account for the missing activity. By building up the model definition in this manner, the user can gain insight into what causes the usage of various resources and where the major impacts on performance arise. This insight is far more important in the planning process than the degree of similitude of the model to reality.

Using the base model, the investigator is now prepared to probe its performance limits, the model analog to stress testing. The simplest way to do this is to increase the specified transaction traffic rates until some resource in the model saturates, usually causing the response time for one or more transactions to "blow up." A special set of rate multipliers representing growth factors is provided for this purpose. Then a careful analysis of the cause of saturation may indicate a simple change in the system to eliminate this potential bottleneck, or it will inform the analyst what system upgrading will be required in the future when the particular growth factor is reached. It will also point out other potential resource bottlenecks that may arise once the primary constriction is relieved. In this way, available system capacity can be explored in terms of growth in several dimensions.

The present model of CICS will highlight the resource-constrained situations, listed in Table 1, for which several causes are given and possible courses of remedial action recommended. (Note that all resource utilization tests are set well below 100 percent. It has been found by experience that most on-line systems do not operate effectively beyond the specified warning levels.)

Now that the investigator has learned where the capacity limits of his current system lie, he can confidently upgrade hardware specifications and add new applications to the model in the form of

Table 1 Resource-constrained situations

Situation	*Cause*	*Remedy*
Processor utilization exceeds 70%	May be due to excessive path length in a particular transaction	Redesign logic to reduce transaction rate or path length
	All path lengths being comparable, the total rate is too high	Upgrade to faster CPU to support the rate
Other CPU bottlenecks	Maximum active task parameter may be set too low	Possibly increase AMXT, although this, in turn, increases the paging rate
	Paging may be causing excessive partition lockout	Investigate reducing paging rate or using faster paging device
	Transient data accesses may be producing excessive partition lockout	Specify multiple buffers or use journaling
Channel utilization exceeds 50%	On-line usage heavy	Use RPS option if available or install a second channel
	Batch usage heavy	Put batch usage on separate channel
DASD utilization exceeds 70%	Small number of units are overloaded	Move active files or split extents onto less active units
	Most units are overloaded	Obtain more storage units or consider faster ones
	May be caused by overloaded channel	See channel remedy above
DASD capacity exceeded	Small number of units are overloaded	Redistribute file allocations, ensuring traffic balance is not upset
	Several units are overloaded	Consider more units; if larger packs, examine the consequence of fewer access paths
Communication line utilization exceeds 70%	Lines are half-duplex	Consider higher-speed lines or switch to full-duplex
	Lines are full-duplex	Consider higher-speed lines or additional lines
Other communication bottlenecks	Response time excessive but utilization is nominal	Investigate pacing parameters; review need for positive response
	Available time between terminal interactions is too small	Add more terminals or improve system response time

new file definitions, new transaction types, and associated processing macros. Once again, he will want to probe the limits of his modified system. With such information at hand, the proposal to implement the new application can include a rational plan for system upgrades in the future. As the anticipated growth materializes and additional applications are added to the system, the

model can grow with it, being used to track the current state of the system and to revise plans to expand the latest system bounds.

Conclusion

The foregoing CICS model is only an example of similar applications-oriented models that have been developed.[9] Various characteristics have been highlighted to show what such an aid should include to be useful in capacity planning.

The design of the model itself must be accurate enough to reflect performance variations that concern the planner but must not require input data that cannot easily be provided. Also, the program implementing the model should be fast, with rapid turnaround, encouraging the user to try a wide range of situations. Only thus will he come to understand the dynamics of his system.

The output of the aid should be crisp and germane to planning needs. Any outstanding system problems should be highlighted in the report, their causes identified if possible, and remedies suggested. Too often, the important numbers are buried in a jumble left from the days of model building, and the user is faced with a dump of "possibly useful" numbers. In short, the use to be made of the results in the planning process should be of paramount concern to the designer of aid output.

Finally, the user should be selective regarding input even when the aid permits minute detail. The input should be simplified as far as possible in order to gain an understanding of the causes and effects of the reported results, which then leads the user to see how to proceed. In capacity planning, it is more important to know where you are going than to know precisely where you are.

If new estimating aids were designed to meet these criteria, and these aids, along with existing ones, were used in an exploratory manner to gain a sense of direction, the productivity of the capacity planner would be greatly enhanced.

ACKNOWLEDGMENT

The author wishes to thank Kuno Roehr of the IBM laboratory at Boeblingen, Germany for clarification of the assumptions behind the batch model.

CITED REFERENCES AND NOTE

1. The model described is ANCICSVS, an IBM Aid only available within IBM.
2. *Customer Information Control System/Virtual Storage, General Information Manual*, GC33-0066, IBM Corporation; available through the local IBM branch office.

3. W. Gordon and G. Newell, "Closed queuing systems with exponential servers," *Operations Research* **15,** 254-265 (1967).
4. J. Buzen, "Computational algorithms for closed queuing networks with exponential servers," *Communications of the ACM* **16,** 527-531 (1973).
5. 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 (May 1975).
6. *Analysis of Some Queuing Models in Real-Time Systems*, Data Processing Techniques Manual, F20-0007, IBM Corporation; available through the local IBM branch office.
7. L. Takacs, *Introduction to the Theory of Queues*, Oxford University Press, Oxford, England (1962), pp. 189-204.
8. W. Chang, "Bulk queue model for computer system analysis," *IBM Journal of Research and Development* **18,** No. 4, 370-372 (July 1974).
9. Y. Bard, "An analytic model of the VM/370 system," *IBM Journal of Research and Development* **22,** No. 5, 498-508 (September 1978).

The author is located at the IBM Data Processing Division Education Center, Bldg. 005, South Road, Poughkeepsie, NY 12602.

Reprint Order No. G321-5116.