Techniques for developing analytic models of computer systems and subsystems relate to establishment of the level of system detail, to selection of significant parameters, to definition of analytic expressions, and to validation of model results.

This paper emphasizes the use of discrete-event models in the development of analytic models, particularly with respect to identification of key parameters and to correlation of results. Described are two analytic models of computer systems, the analytic techniques employed, their relationship to corresponding discrete-event models and their advantages as performance evaluation tools.

Techniques for developing analytic models

by A. L. Anthony and H. K. Watson

Simulation models are recognized as valuable tools for the performance evaluation of complex computer systems. Such models are made to represent systems at various levels of detail depending upon their application and required accuracy. The two most often-used techniques are discrete (stochastic) modeling, where events flow in discrete intervals of time through representations of system logic, and analytic modeling, where gross system performance is described in terms of mathematical equations.

Those who create simulation models for system evaluation tend to include levels of detail approaching that of the system being modeled. Perhaps this tendency is due to a fear that some significant detail may be omitted. Or, perhaps the languages that are available for discrete system simulation make it so easy to include these details that one begins to overlook the penalties that he must pay (execution time, preparation time, assembly time, etc.).

Let us state at the outset that there are applications where very detailed simulation models are needed, such as for early system design and development. For example, such models may be used for assistance in selecting one of several alternative algorithms or in analyzing logic paths where timing is critical. How-

ever, models used for evaluation of total system criteria such as throughput, response time, and component utilizations do not require this detail.

Our experience has shown that analytic modeling techniques provide an excellent capability for evaluation of gross system criteria. Furthermore, analytic models use less computer time, provide faster turnaround, and are easier to set up and modify than their discrete system model counterparts.

Development of analytic models of computer systems requires identification of significant parameters and of relationships between these parameters. If discrete models are available from the design phase of system development, this information can be obtained for the construction of a representative analytic model. Knowledge of the relationships of parameters helps to establish the technique for describing analytic models, be it purely algebraic, empirical, or queuing theory.

This paper presents some techniques that have been used in developing analytic models using information learned from discrete models. Brief definitions are given of each type of model and advantages are shown of analytic techniques in computer system performance evaluation. Two analytic models of the IBM System/370 Model 165 are used to illustrate these techniques. Finally, we present a general discussion of the application of these techniques to other simulation studies.

Simulation

discrete-event modeling

Discrete-event simulation implies that a change in the state of the system takes place only when an event occurs at some distinct instant in time.[1] In a discrete-event model of a computer system, for example, an event such as the completion of the processing time for a particular function, the initiation of a read or write access to an I/O device, or the placement of a message in a queue would cause a change in system state.

A discrete-event model monitors the interactions of events and records statistics about the occurrence of significant events. Execution of such a model must be continued long enough to achieve sufficient sampling of events over a period of time during which the system is relatively stable. In most cases, the model must be run through a period of initialization to stabilize the system before measurements can begin. It is usually necessary to periodically inspect the state of the system during the initialization period to determine when the system has reached a stable condition. The time required to achieve this stable status contributes significantly to the total time of a simulation run.

Considerable effort has been expended in the development of higher-level simulation languages to support the design and implementation of discrete-event models, and these languages have proved valuable. The General Purpose Systems Simulator (GPSS),[2] Computer Systems Simulator (CSS),[3] and the SIMSCRIPT II Programming Language[1] are examples of this type of simulation language.

analytic modeling

Analytic simulation implies that all changes may occur at all times for all states of the system. In an analytic model, then, all system activity is occurring at all times as opposed to the discrete-event model where any process proceeds sequentially through its activity.[4]

The continuous process is represented by sets of equations that are solved for the average state of the system. A computer system, for example, may be represented in an analytic model by equations which express system characteristics such as service times, queues, and probabilities of events. For an analytic model, a set of system parameters is defined using specified average values and distributions; since solvable analytic equations represent only approximations of real computer systems, satisfactory representations by analytic models may be more difficult to achieve and more difficult to understand than those of equivalent discrete-event models.

Validation of the model must be an inherent part of the development process. Validation implies the comparison of simulation results to measurements of the real system, to another model known to be accurate, or to some adequate criteria to ensure that the model is producing accurate data. If these comparisons indicate that the analytic model results are not sufficiently accurate, corrections are applied to the model and the procedure is repeated. This validation process generally involves several iterations of model changes and system measurements before a satisfactory confidence level is achieved.

No initialization period is required of analytic model runs because the equations are solved directly for the average state of the system. However, if there are several unknown parameters in the model, the solution of sets of equations may require many iterations to determine a common set of values that satisfies the input conditions, that is, to determine the point where the solutions for all unknown values intersect. When these solutions have been found, the model is said to have converged. It is possible for a model to fail to converge, but experience has shown that given a well-designed analytic model, failure to converge is invariably due either to an incompatible set of input parameters or to a system that is hopelessly overdriven, with component utilizations exceeding one hundred percent.

General purpose higher-level programming languages such as APL[5] and FORTRAN[6] provide excellent vehicles for expressing analytic models.

Reasons for developing analytic models

Analytic models are preferable because their implementation is usually more efficient than that of discrete-event models. Execution time is less because no initialization period is required, procedures are less complex, and their languages execute more efficiently. An application language such as APL also provides the user with interactive communication with the simulation model. This capability expedites modifications and reduces set-up and turnaround time. These efficiencies result in analytic models that are less costly than their equivalent discrete-event counterparts.

Analytic models are particularly adaptable to the evaluation of computer systems when emphasis is placed upon gross performance characteristics such as component utilizations, job throughput, service times, terminal response times, and subsystem queuing. The discrete-event model is applied when greater detail is required to analyze the internal relationships within components of the total system. Measurements and results from the detailed model can be used to expedite the development and to provide information needed for verification of the analytic model. Types of system evaluation best performed by analytic models are configuration analysis to evaluate new components, utilization studies to establish system balance, and load studies to determine job throughput and system response times.

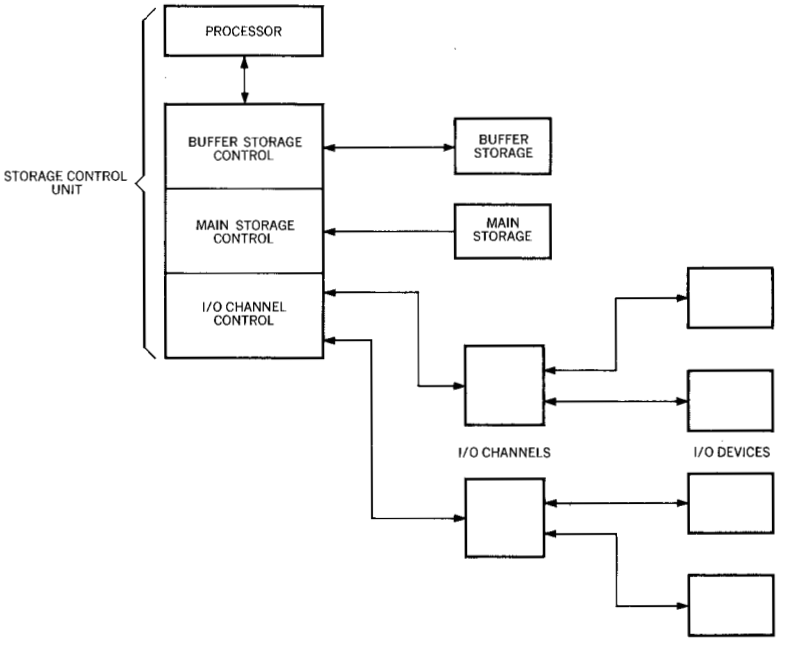
Another reason for developing an analytic model is to provide a supplement to the discrete-event model that expedites the gathering of performance data during an extensive series of simulation runs. We recently used an analytic model for this purpose. In the procedure that was followed, first the analytic model was calibrated using the results from a few selected runs of the discrete-event model. Then the analytic model was used to obtain performance data both within the range of calibration and beyond this range by extrapolation. Another application of the analytic model provided a guideline to significantly reduce the number of simulation runs by the discrete-event model. The value of the analytic model becomes apparent in comparing its processing time of less than a minute with a time of 30 minutes to three hours for the discrete-event model.

Development of an analytic model

PRAM/ORAM models

Two analytic models were developed for the IBM System/370 Model 165: the Processing Rate Analytic Model (PRAM) and the

Figure 1 Schematic of a buffered storage system



Overrun Analytic Model (ORAM). The PRAM model is used to determine the instruction execution rate attained by the processor for any specified instruction stream and I/O data rate. The ORAM model is used to determine the probability of overrun for each channel of any specified configuration. Both models were developed as a result of the experience gained from the discrete-event simulation of the IBM System/370 Model 165. This simulator was programmed in the GPSS V language.[2]

description of discrete-event simulator

The discrete-event simulator represents the major components of a computer system: the central processing unit, storage, storage control unit, I/O channels, and devices (see Figure 1). Discrete events are the execution of an instruction stream derived statistically from a real program in the processor, references to main storage for instructions and operands, and data transfer between I/O channels and main storage. Control functions are simulated in the storage control unit, resolving the priorities of requests to storage.

Instruction execution by the processor is represented by a specified instruction mix, stated in terms of a distribution of classes of instructions and a distribution of instruction execution times for each class. Simulation of an instruction includes a random class selection, a processing time designation and assignment of the number of references to storage. The system contains a two-level, hierarchical storage consisting of small, fast, buffer storage and a large, slower-speed, main storage. If a storage request

cannot be satisfied in the buffer storage, a block of data containing the required address is fetched from main storage and placed in the buffer storage (replacing a block of buffer storage that had not been recently used).

The storage control unit is represented by its three subunits: I/O channel control, buffer storage control, and main storage control. Simulation of this unit controls the contending processor and channel requests for main storage, resolves the access to buffer storage, and initiates storage cycles according to stated priorities of requests.

The discrete-event model is used for:

- Evaluation of processor performance
- I/O loading studies (configuration analysis)
- Channel data rate capability (overruns)
- Design analysis of selector channels and storage control unit
- Evaluation of priority schemes
- Optimization of device latency
- Determining utilizations of processors and devices
- Queuing analysis

buffered storage analytic representation

Central processor performance is expressed in terms of its instruction execution rate (MIPS, or millions of instructions processed per second). The discrete-event model of the IBM System/370 Model 165 showed that attained MIPS is primarily dependent upon the following:

- Probability of buffer hits
- I/O data rates and types of channels
- Instruction mix and its execution sequence
- Interference between the processor and I/O
- Interference in the processor itself
- Processor and storage cycle times
- Task-switching rates (a task switch invalidates information in the buffer storage)

Sets of curves drawn from results of simulating the system with this model showed that changes in system parameters tended to cause exponential changes in MIPS. Evaluation of these results led to the development of an expression for the instruction execution rate of a processor with respect to storage references. The initial expression for MIPS was

$$\text{MIPS} = Ae^{kx} \tag{1}$$

where x had the form

$$x = BP^{a}_{1} + CP^{b}_{2} + DP^{c}_{3} + EP^{d}_{1}P^{f}_{2} + FP^{g}_{3}P^{h}_{1} + GP^{j}_{2}P^{k}_{3} + HP^{l}_{1}P^{m}_{2}P^{n}_{3} \tag{2}$$

with
P_1 = probability of a buffer hit
P_2 = probability a store operand is in buffer
P_3 = probability of other references to the buffer

The constants in Expressions 1 and 2 were derived and evaluated for both the IBM System/370 Model 165 and the IBM System/360 Model 85. A least-squares curve-fitting process was used with values of MIPS obtained from the hardware simulators. The resulting expression provided instruction execution rates that were within five percent of values provided by the hardware simulators. However, this approach was abandoned because it became too difficult to evaluate the constants, it was almost impossible to find meaningful relationships between sets of constants for different types of systems, and we never really understood why the constants took on the specific values they assumed.

This investigation did establish the fact that MIPS can be expressed as an analytic function. The evaluation of this expression resulted in an expansion of the concept through the development of a model based on key hardware and software parameters, and this proved to be a working model.

development of PRAM

We will now discuss the development of the equations for PRAM based upon the knowledge gained from the discrete-event model. The analytic techniques of PRAM are a combination of classical queuing theory and empirical relationships of system parameters pertaining to requests for main storage.

A close look at the results from the discrete-event model then led to the definition of a set of key parameters that affect MIPS. This parameter set included the following:

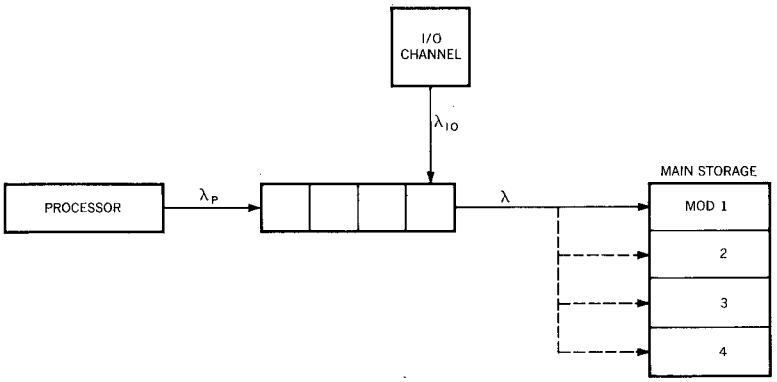
- Instruction mix definition
- I/O data rate
- Main storage cycle time
- Number of storage requests per instruction
- Main storage interleave factor
- Buffer storage probabilities

These parameters were combined logically into a set of algebraic equations related to the basic equation:

$$\text{MIPS} = \frac{1}{t + U(acc + wq)} \tag{3}$$

where
t = average instruction execution time with no main storage requests
U = fraction of references that delay the processor

Figure 2 Main storage queuing model



acc = average access time to main storage
wq = average waiting time in queue for storage[7]

$$wq = \frac{\rho}{(\mu - \lambda_{IO})(1-\rho)} \times \frac{1 - N\rho^{N-1} + (N-1)\rho^{N}}{(1-\rho^{N})} \qquad (4)$$

where
ρ = λ/μ = main storage utilization
μ = service rate of main storage
λ = average request rate to main storage
λ_{IO} = average I/O request rate
N = maximum number of requests allowed in the queue

The entire set of equations represents the queuing model shown in Figure 2.

The first attempts at validating this model (correlating with results from the discrete-event model) were made without any exponential factors. Validation results were reasonable but were not as accurate as desired. Further experimentation showed that fine tuning of the equations could be achieved with the inclusion of certain empirical relationships. The following factors were then introduced into the above equations:

$$A = \rho e^{1-\rho}$$

$$B = \lambda e^{1-\lambda}$$

$$C = \sqrt{A \times B}$$

Thus

$$U = A\Big(s + P\ (\lambda_{IO} + r - s)\Big)$$

where
s = average number of stores per instruction
r = average number of main storage requests per instruction
P = probability that a main storage fetch causes processor delay

and

$$N = 1 + 3C$$

The factor A was used to modify the effects of the I/O data rate and of U in Equation 3 above. N of Equation 4 was allowed to take on nonintegral values with modification by the factor C. With these modifications, good agreement was obtained not only for the Models 165 and 85, but also for several other systems from which measurement data was available.

The relationships between equations of the PRAM model imply an iterative procedure for the calculations of MIPS, starting with an assumed value for the average request rate (λ) to main storage. Iterations continue through the equations until the value of λ converges within a specified limit.

In most cases, the value of λ converges in a few iterations. An innovation is used by PRAM when the convergence of λ causes the queuing equation for wq to become unstable. Minor adjustments are made to the value of N in an attempt to achieve stability of the queuing equations and to compute a value of MIPS that otherwise could not be determined. This adjustment is limited to ensure that results remain within desired accuracy.

development of ORAM

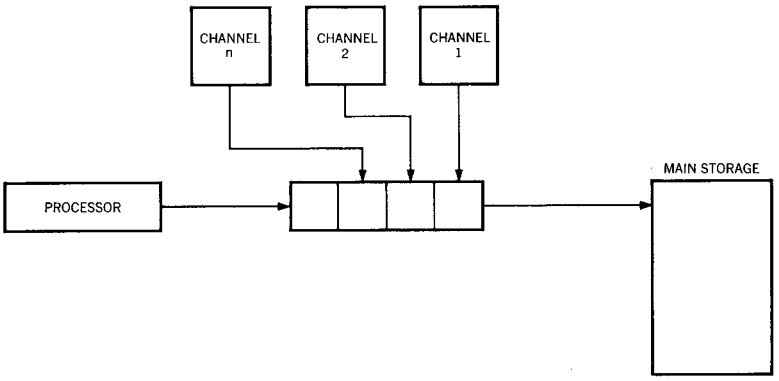
The ORAM model simulates I/O channel configurations serving storage requests of devices with constant data rates. Each device must be served in the time period required for the transfer of a specified number of bytes of data. The device is then ready to initiate another storage request. If a storage request is not satisfied within the specified time period, data is lost, and an "overrun" is said to occur.

The net result of an overrun is lost time only, since the channel and device then must be resynchronized (or reselected) before data transmission can continue. It also may be necessary to go back to the start of a block of information and retransmit part or all of the data that had previously been transferred. Since system control programs provide the capability of recovering from an overrun, an occasional such occurrence is not serious. However, frequent overruns can result in significant loss of time, so systems are configured to minimize the overrun probability.

The probability of an overrun depends upon the data rates of all channels in the system, the priority order of the channels, and other system parameters such as storage cycle and channel timings.

The discrete-event model determines the exposure of any configuration to overrun. However, determining the absolute probability of overrun for any channel is expensive because of the

Figure 3 The ORAM analytic model



computer time needed to obtain a significant sample of events (as much as two hours of IBM System/360 Model 75 time).

ORAM is a very simple analytic model. The maximum (worst-case) service time for each channel is calculated by summing the service times for the channel itself and the memory interference delays (storage cycle times) that can be caused by channels of higher priority and one channel (or the processor) of lower priority. This worst-case time is then compared with the time between requests for a specific channel to determine whether or not that channel could overrun. If a channel can overrun, the probability of the events that could result in overrun is calculated from the binomial probability distribution. This then, is the probability of overrun for the channel. Figure 3 shows this model.

In developing the ORAM model, the logic paths of the discrete-event model were traced, and algebraic equations were written to calculate the channel service time. We determined from the discrete-event model that this service time could be represented in terms of three channel-time parameters plus the storage access time. To this basic channel service time was added the additional time due to interference from the other channels. One lower priority channel (or the processor) and each higher priority channel might delay access to storage by one storage cycle time.

The discrete-event model showed that overrun probability is a function of the data rates of all channels sharing access to a common main storage; that is, it is a result of concurrent requests for storage from several channels. Therefore, we wrote equations to calculate the probability of all combinations of requests that would result in interference long enough to cause overrun.

Several experiments were then made with the analytic model, using configurations that had previously been simulated with the

discrete-event model. The analytic model in every case correctly distinguished between configurations that could and could not overrun (zero overrun probability). Accurate overrun probabilities were obtained for those configurations where the chance of overrun was small. However, the model results were consistently lower than those provided by the discrete-event model for configurations with high data rates.

Consequently, additional discrete-event simulation runs were made using configurations with higher data rates than the real system was designed to handle. We found that higher-priority channels with high data rates were requesting access to storage more than once before a low-priority channel was served. After the equations were modified to consider this possibility, accurate worst-case service times were obtained for these configurations. Additional effort was then spent in modifying the overrun probability equations so the results from the two models agreed within five percent for configurations that could overrun.

summary of techniques

The recommended approach to good modeling, then, is to establish a level of detail sufficiently gross to keep the model simple yet specific enough to produce accurate results. In most instances, average values of parameters are satisfactory. Where distributions are required, consideration should be given to development of a discrete-event model of critical components of the system. If possible, the discrete-event model should be validated by measurement of the real system or by detailed analysis of flow diagrams and significant events.

Experiments with a discrete-event model are used to identify parameters which are most critical to the system performance. Timing information, data flow, load conditions, and performance characteristics are also obtained from such a model. Experience with the discrete-event model then is applied to the development of an analytic model of the system by first determining the information that is needed from the discrete-event model and then attempting to establish mathematical relationships between this information and that which is already known about the system.

Computer system performance is expressed in terms of task throughput, terminal response times, and device utilizations with respect to given loads and system configurations. Associated performance criteria are processing rates, queuing characteristics, service times, and capacity. The relationships of these performance parameters can be established in an analytic model to produce results that are satisfactory for functional specifications, configuration analysis, and function tradeoffs.

Analytic models include such techniques as simple algebraic equations, probability theory, and queuing theory. Algebraic

equations can be used to express simple relationships such as the summation of component parts, the inverse relationships of request rates to service times, and the distributions of events. Probability theory can be used in the representation of distributions and decisions as well as in the expression of those combinations of events that are likely to occur. Queuing theory presents the most complex representation of the three techniques discussed in this report. Queuing equations relate characteristics such as request rates, service times, number of servers, priorities of requests, preempting of service, and queue lengths. Computer systems exhibit all these characteristics.

The final phase of model development is its validation (verification that results are reasonable). Techniques used in the validation of an analytic model are

1. Validation of parameters and results with measurement of the real system
2. Cross correlation of its results with those of a discrete simulation model
3. Some combination of measurement and discrete modeling

The most accurate method of validation of a model is obtained by measurement of the real system. This technique requires the availability of either the system itself or an early prototype. However, the fact that a system exists should not discount the utility of an analytic model of that system. Such a model is very useful in the study of new components, new configurations, and system loads.

The validation process should be limited to the range of parameter values applicable to the normal operating conditions of the system. First, acceptable tolerances must be established. Then emphasis should be placed upon the critical parameters—those that have the greatest effect on performance of the system. Reasonable assumptions are satisfactory for the less critical parameters.

Most validation processes require iterations of model runs and measurements. In using this procedure, it may be necessary to introduce empirical relationships into the analytic model to achieve proper correlation with the discrete-event model. The procedure can employ a curve-fitting process. Equivalent plots of performance criteria can be observed for correlation. If the curves agree within the allowable tolerances over the normal range of operating conditions, the analytic model may be considered as validated. During this process, redefinition of some of the equations may be necessary. It may also be necessary to reevaluate the basic theoretical assumptions, to introduce empirical relationships of parameters, to fine tune some of the parame-

ter modifiers, or to reconsider which factors are critical with respect to performance. Once validated, the analytic model can be employed with a high confidence factor in the evaluation of the real system.

Summary

The development of analytic models is facilitated by the availability of related discrete-event models. Validation of analytic model results can be achieved by correlation with results from the discrete-event models. Together these models provide complementary vehicles for the evaluation of system performance at various levels of system detail.

CITED REFERENCES

1. P. J. Kiviat, R. Villaneuva, and H. M. Markowitz, *The SIMSCRIPT II Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1968).
2. *General Purpose Simulation System V*, Program Number 5734-XS2, IBM Corporation, Data Processing Division, White Plains, New York.
3. *Computer System Simulator II*, Program Number 5734-XS5, IBM Corporation, Data Processing Division, White Plains, New York.
4. D. A. Fahrland, "Combined discrete-event continuous systems simulation," *Simulation* (February 1970).
5. *APL\360 User's Manual*, No. GH20-0638, IBM Corporation, Data Processing Division, White Plains, New York.
6. *FORTRAN IV Language*, No. C28-6515-7, IBM Corporation, Data Processing Division, White Plains, New York.
7. P. M. Morse, *Queues, Inventories and Maintenance*, John Wiley & Sons, Inc., New York, New York, 18 (1958).