A trace-driven modeling technique for computer system evaluation is discussed. This approach intends to solve the inherent dilemma in computer systems modeling of too many simplifying assumptions or of too much detail.

An experimental model based on this technique is also discussed. The objective of the model is to study the effect on performance of various multiprogramming and multiprocessing system design choices. Model implementation is focused on those aspects of the system contributing substantially to total system performance. While the operating system is conceptually modeled, detailed logic and timing are supplied in the job-trace profile, reducing modeling effort and improving model flexibility.

Trace-driven system modeling

by P. S. Cheng

Although electronic computers have been in use for only about twenty years, they are already in their third generation. Computer systems have grown both in size and speed, and computer system applications have become exceedingly complicated. The number of significant variables in a present-day system is large and their interrelationships are complex. Because of the burgeoning of new equipment and the striking changes in programming support, numerous problems are confronting computer systems manufacturers and users in evaluating the effectiveness of both equipment and operating system configurations for a given workload. However, few guides are available to aid in solving these problems. System evaluation tools and techniques are more urgently needed than ever before, both before and after system design and system installation.

Of course, if the situation permits, the best way to obtain quantitative data is to run the system and monitor the relevant events.[1] Unfortunately, this ideal situation seldom exists. Instead, one of two alternative approaches is generally used in attacking the evaluation problem.

The first approach is involved with simple algebraic, statistical, or algorithmic analysis applied to those sophisticated and analytical techniques such as building a mathematical model based on Markovian or probabilistic methods to meet a certain specific evaluation goal. It has been proved by several models[2–5] that results are quite satisfactory even for relatively complex situa-

tions. However, such models usually have rather limited applicability and lack the flexibility to permit a number of different system or algorithmic modifications to be imvestigated with a minimum of additional effort.[6] Moreover, mathematics itself is subject to limitations, and a mathematician may be unable to construct a model of a fairly complex system so as to obtain a desired result.

The other alternative is simulation, in which experiments are performed on a model that is supposed to match at least a set of relevant characteristics of a real system. Although its history is brief, simulation is becoming increasingly popular in many disciplines. However, much room remains for improvement in the practice of the technique.[7]

In general, system simulation is accomplished by building a model in a time domain. The flow of control and of simulated data through the model is similar to that in a real system. Because programming support is considered an integral part of a system, a computer system configuration is modeled by specifying the equipment, the supporting programs, and the interactions between them. Among the basic equipment that must be simulated are: the central processing unit (CPU), main storage, channels, control units, disk and drum storage devices, tape drives, and unit record equipment. These components have characteristics, such as data transfer rates, access times, and seek times, which must be reflected in the model. The modeler must also specify the various scheduling and dispatching algorithms, data management characteristics, interruption-handling disciplines, and other operating-system functions.

Another basic requirement for computer system simulation is the ability to specify formally the expected job mix and constraints under which the simulated system must operate. Although there are some higher-order, special-purpose languages[8] to relieve the modeler of most of the burden, the simulation language itself provides only the modeling facilities. The model must still be constructed. And to do this, the modeler must be familiar with the operating system and the job mix, so that he can represent them in the simulation language.

For complex systems, such as those providing multiprogramming and multiprocessing, it is generally not possible to know completely all of the factors, or parameters, affecting the behavior of the real system. Some simplification is necessary, and only the parameters strongly affecting certain preselected areas are considered. In an extensive system, the number of parameters involved becomes quite large; in order to stipulate some of the free parameters, some parameters must be either ignored or estimated indirectly as functions of more accessible ones. Aside from the loss of flexibility and of predictive precision, this parameter-selection process in itself is a painful task with no guarantee of success. Perhaps one source of trouble is starting with assumptions that are too vague; another source may be the introduction of

pseudo-functions, i.e., functions that can be checked only by limited individual cases.

Another approach to making simplifying assumptions—detailed modeling by faithfully representing in a simulation language both the real system and a simulated job stream—is often so time-consuming and costly that it is impractical. Using this approach, the modeler imitates the system as it is coded, with little provision for altering basic program modules. The resulting model approaches the complexity of the real system. It is not very useful for either demonstrating the principles or for revealing the unnoticed interactions among various parts of the system.

To overcome these problems of modeling, a lesser-known technique called "trace-driven modeling" is discussed in this paper. In the trace-driven approach, data traced on a real, running system are used to drive the model. The workload and the activities of system components in response to the workload are supplied as input to the model in the form of trace data. The trace data are obtained dynamically by monitoring significant events while the operating system and the workload interact.

Trace-driven simulation eliminates the tedious work of either specifying the relevant system parameters or coding in detail the operating system and the workload in simulation language. (Gross models of some portions of the operating system are required.) This approach also eliminates the need for detailed knowledge of the workloads.

In this paper, the discussion of trace-driven modeling includes the concepts of the method with a description of the trace data. The use of the trace-driven approach for a particular model indicates its value in determining system performance.

Trace-driven modeling

In computer system modeling, system behavior is governed by certain characteristic parameters. Ideally, parameter values should be measured in a real system to fully realize the predictive capability of modeling. In addition, models should be tested with data from the real world. This is done by trace-driven modeling.

primary requirement

The primary requirement for trace-driven simulation is to run the job stream sequentially on a basic equipment configuration, using an operating system under which a trace program can be executed. System activities, including CPU processing and input/output operations that pertain to the execution of a given job, are monitored. Monitoring is the recording of all the system functions required to process the work load. Because of the modular construction of the operating system, its parts are or are not executed depending on whether or not the work load requires the function. Consequently, a relatively simple job profile can be created.

Given this kind of trace data, the only remaining items that we must consider are the criteria used in the system to allocate the resources needed for the workload. For example, the simulated system would reflect the scheduling algorithm, task dispatching and switching rules, input/output queuing principles, and interruption handling. The models would also reflect characteristics of the new system, such as CPU speed and the number of input/output channels. The simulated system makes some decisions dynamically, depending on the state of the system, just as though the job stream were being executed in a real system.

data-flow structure

From another perspective, the model can be looked upon as a data-flow structure, in which data flows among system components, queues build up, and contention patterns establish themselves. Job queues, task queues, job scheduling, and resource allocation are modeled. The equipment configuration and characteristics are specified as system parameters to be used as input to the model.

The overall data traffic pattern within the system is controlled in accordance with the traced data that is supplied to the model as well as with the system algorithms. The trace data accounts for all the system times, broken down. All input/output activities and CPU processing time (spent, for example, in executing problem programs or in performing such system services as compiling) is recorded; the time not spent in CPU processing is tallied either as a wait for input/output completion or as idle time. By tracing system activities in this manner, nearly all logical input/output operations can be identified and associated with specific data sets. So long as there is a minimum of operator intervention and no equipment malfunctions to introduce uncontrolled variations, the trace data can be considered as a faithful reproduction of the actual workload in a static form. A sample of a traced record format[9] is shown in Figure 1.

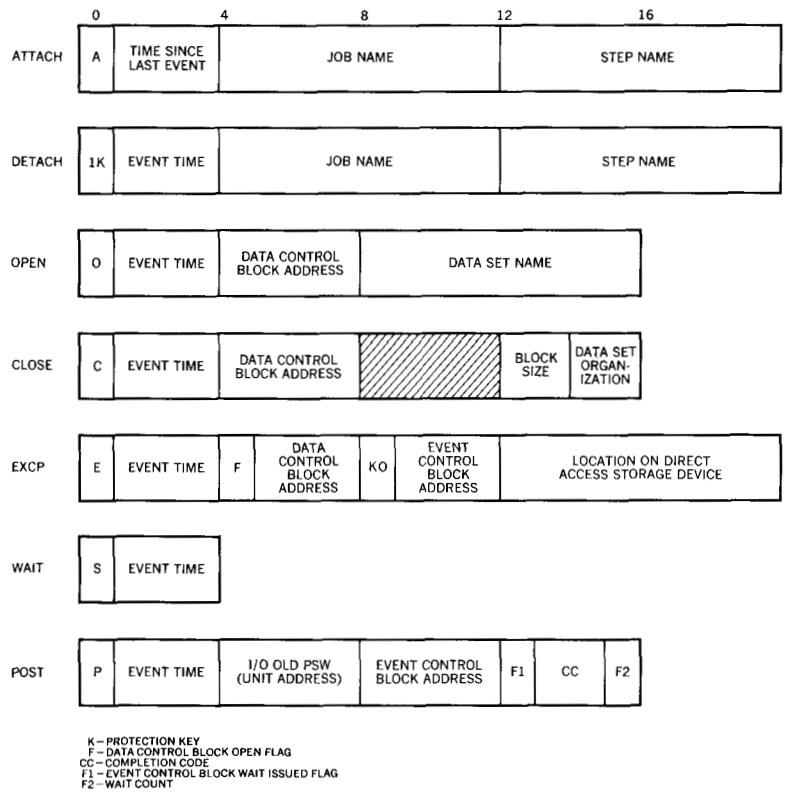
model characteristics

In evaluating computing systems through simulation, a modeler's motivation determines what information and level of detail is of interest. Thus, an ideal model should have these dual characteristics:[10] the ability to simulate a wide variety of equipment and operating system configurations without extensive remodeling, and the ability to simulate different parts of the configuration at different levels of specificity.

Sometimes it is difficult to build a model with these dual capabilities, partly because of the need for making simplifying assumptions. The modeler must decide which real-world attributes to incorporate into the model. At a later time, other attributes of the system or greater detail may be needed in the model.

A simulation model is arbitrary and can be anything its creator desires it to be.[11] The elements of the system configuration to be modeled are similar to the modules of the operating system in that each performs a certain function. These elements can be considered as basic "building blocks" in the trace-driven model. Such blocks can be combined in a variety of ways so that many

Figure 1 Sample traced record format



representations of a system can be modeled as long as the relationship among these elements can be identified. Thus, although the trace data are obtained using a given configuration, it can be used to drive a model of a different system. Even changes in the programming system can be mapped into the new configuration as it is simulated by extrapolation or projection.

Simulation model

The trace-driven approach was used in an experiment directed toward the determination of gross system performance as various multiprogramming and multiprocessing system configuration changes were made. Each configuration presented a unique set of processing requirements. For example, the target system may have changed in hardware components, the arrangement of the components, the operating system, or the job mix. However, to study computer system organizations like these, the major interest is not in these elements by themselves but in the complex interactions among them that cause changes in the performance of multiprogramming and multiprocessing systems.

In this simulation model, we assume that execution of a given job involves some system functions together with the job's own computing and input/output demands. The magnitude of these requirements may vary from system to system, but the functional requirements are independent of the equipment configuration on which the job is performed. Model implementation is thus focused on the aspects of a total system that substantially contribute to system performance. The emphasis is on scheduling, dispatching, resource allocation, and the effect of changing the operating environment, that is, the equipment and the procedures for interaction between devices.

input data

Two sets of data are supplied to the model as input. The first is the simulated target system configuration specified to the model for each run; the other is supplied to the model for each set of jobs in the form of trace data that contains the job profiles resulting from executing the jobs in an existing system. In order to eliminate from the trace data any of the effects of multiprogramming, the trace is performed while the job stream is being executed sequentially on a known configuration under a known operating system. Job mix profiles produced in this manner are more nearly independent of the system in which they are obtained. Nevertheless, trace data obtained in this way can be used in the simulated multiprogramming system, where tasks compete for system resources.

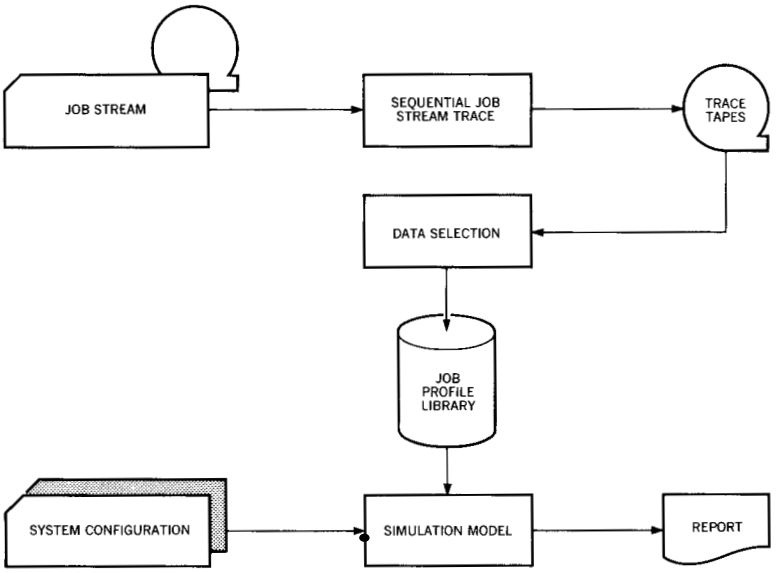
Each job is segregated into one or more job steps, and each job step is represented as a collection of computing segments. Within a computing segment, CPU processing time is coupled with some associated input/output activity. The CPU processing time is further broken down into a combination of CPU processing that is and that is not overlapped with its own associated input/output operations. Run time of a job executed sequentially is dependent upon the number and duration of these CPU processing times and their associated input/output activities. In a multiprogramming system, throughput is further affected by interaction with the other jobs being processed concurrently.

implementation

The implementation of a trace-driven model involves three distinct phases: tracing a profile, selecting data, and simulating activity. The flow of data among these phases is illustrated in Figure 2.

The program for tracing a profile records computing segments as previously mentioned. All requests for transmission of data to and from input/output devices and the corresponding posting of completion of input/output events are captured on a real-time basis and recorded in chronological order. Although the jobs are executed sequentially as they are monitored, the tracing is achieved by using the multiprogramming facility provided by the IBM SYSTEM/360 Operating System. Other techniques used in the monitoring activity rely on characteristics of IBM SYSTEM/360 computers. The interface of the trace program with the operating system is established by the new program status words (PSW);

Figure 2 Overall flow



control is passed to the trace program upon the occurrence of a CPU interruption.

In addition to input/output events, some system functions are monitored either by use of an interruption or by the supervisor-call (SVC) instruction. For each occurrence of these events, a record is created. While timing is read from the internal clock and recorded; the related job step or data set name is preserved and subsequently passed to the second phase (data selection) for further analysis. Figure 1 shows such a trace.

data selection

Data selection involves reducing the data captured during the dynamic trace, so as to reduce the volume of data processed by the model, and selecting the data needed for the particular simulation. This compression is achieved with the aid of utility programs that correlate, for example, a request for data transmission and the posting of completion of the input/output event into a single event. As a result of this process, job steps are identified and segregated. Data set names, together with their associated block sizes and disk or drum addresses, are resolved, if applicable. The information extracted from the dynamic trace represents the system load on the original configuration and consequently is mapped into the new configuration to be simulated by the model.

Another function of this phase is to prepare the trace data for the particular system to be modeled. For example, a detailed profile of events would not be usable in a gross system model. The data selection programs are designed as supporting programs

to be run periodically with the model as the demands of the simulated systems require. In our case, the data compression for each set of jobs is a one-time operation designed to separate the time-consuming tracing and data-reduction activities from the modeling development process. Upon completion of this operation, data in the reduced form are kept on a direct-access device in a job profile library. Sometimes it may be desirable to measure the effect of changes other than those to the system configuration; for example, the effect of a data set organizational change might be of interest. Such changes can usually be done in this phase simply by rearranging or reorienting the data-set characteristics. Under such circumstances, short periodic runs may be necessary to prepare the input to the model for various requirements depending on the simulated conditions.

local autonomy

Design of the model is based on a local autonomy principle that is used in the operating system for optionally allocating its resources. The model is made up of several units called subroutines, each of which assumes some control, operational, or housekeeping function, or a combination of these functions. These subroutines can be considered as subsystems of the total system. Interfaces are established via event flows among them. Whereas details needed in each such subroutine depend on what is being investigated, the local autonomy permits an individual subroutine to make decisions in its simulated environment that is within its jurisdiction. When an event occurs, appropriate action takes place in one or more subroutines to reflect the change of status. The combination of these actions represents the full functional capability of the operating system. The need for detailed modeling is thereby sharply reduced, and coding and model-processing time is correspondingly minimized.

The model was implemented in an event-based language.[12] In this context, event suggests a relatively significant activity, such as the transfer of data from main storage to an input/output device, rather than the execution of a machine instruction. It should also be noted that the occurrence of an event may sometimes change the status of the simulated system, causing the occurrence of one or more other events. Throughout the model, the duration of an event is modeled with two occurrences, one at the beginning and one at the end of that period during which the activity is taking place.

In the job profile, the traced job data that are required to drive the model are ordered into events, and all necessary information and timing are preserved as attributes in an event parameter list. Once the system configuration has been specified and resources allocated, the primary function of the model is to time these CPU-input/output events in accordance with the scheduling and dispatching procedures built into the model. The traced CPU processing times in the CPU-input/output segments are either expanded or contracted in conformity with a computing factor specified for the simulated system. Logic and algorithms are

provided in the model to issue the simulated input/output request and to compute the related device's access, seek, rotational delay, and data transmission times. In the simulated system, jobs are initiated, executed, and terminated according to the availability of resources and operational specifications for the simulation run. All job step starts, stops, and elapsed times are tabulated and accumulated in the total simulated processing time for all jobs within the workload.

In addition to reporting the simulated throughput and elapsed time for all jobs in the specified workload at job-step level, the model collects and reports device utilization for all CPU's, main storages, input/output devices, control units, and channels. Data set reference statistics and information on all system queues and components are also included in the output at the end of a simulation run.

Prior to the simulation, the model requires just about the same type of information that a system analyst must consider: given a specific equipment configuration, operating system, and job mix, in what manner should the application be organized on this system configuration? Invariably the system analyst is faced with a variety of tradeoffs in allocating the resources to the requirements of the job mixes. The model requires a similar allocation of resources.

Summary comment

This paper has attempted to introduce the basic concept of trace-driven modeling. As in all simulation approaches, one major concern is the precision of predicitive capability. The trace-driven approach lends itself to accurate predictive evaluations by successive calibration steps. Given an existing configuration capable of executing a given job mix in an operating-system environment, the time required to run the jobs can be measured and firmly established. On the other hand, the job profiles for this set of jobs can also be obtained using a trace program under the same conditions. The output from the model that runs with the parameters representing the physical system can be compared directly with the measured results. Because of this inherent property, accuracy of the model can be established prior to its use on a configuration that cannot be measured.

In this paper, the model was presented as a demonstration of trace-driven modeling. Although no formal effort was conducted toward validation of the model, results from a preliminary investigation were satisfactory. The experience gained from this experiment has convinced us that the trace-driven approach is feasible for computing-system evaluation. In computing science, as in other scientific areas, changes generally occur in an evolutionary fashion. Therefore, the trace-driven technique may well be applicable for future system evaluations.

ACKNOWLEDGMENTS

The trace-driven model development is a composite of the contributions of a number of individuals. Special credit goes to L. Corbet and D. I. Jones for their work on trace-driven simulation and the development of the trace program. The author is indebted to D. M. Braddock and C. B. Dowling for their design and development of the SEAL language and their many contributions to the trace-driven modeling approach, and to J. C. Gibson and W. M. Kochant for their early insight into multiprogramming, tracing, and simulation.

CITED REFERENCES AND FOOTNOTES

1. C. T. Apple, "The program monitor—a device for program performance measurement," *Proceedings of the 20th National Conference of the Association for Computing Machinery* **P-65**, 66–75 (1965).
2. A. L. Scherr, *An Analysis of Time-Shared Computer Systems*, MIT Press, Cambridge, Massachusetts (1967).
3. J. L. Smith, "An analysis of time-sharing computer systems using Markov models," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **34**, 87–95, (1969).
4. D. P. Gaver, Jr., "Probability models for multiprogramming computer systems," *Journal of the Association for Computing Machinery* **14**, No. 3, (July 1967).
5. J. D. Foley, "A Markovian model of the University of Michigan executive system," *Communications of the ACM* **10**, No. 9, 584–588 (September 1967).
6. N. R. Nielsen, "The simulation of time-sharing systems," *Communications of the ACM* **10**, No. 7, 397–412 (July 1967).
7. R. W. Conway, "Some tactical problems in digital simulation," *Management Science* **10**, No. 1, 47–61 (October, 1963).
8. P. H. Seaman and R. C. Soucy, "Simulating operating systems," in this issue.
9. The terms used in Figure 1 are explained in Parts II and III of the article, "The functional structure of OS/360:" B. I. Witt, "Job and task management," and W. A. Clark, "Data management," *IBM Systems Journal* **5**, No. 1, 12–51 (1966). Further explanations are found in the articles by J. W. Havender, "Avoiding deadlock in multitasking systems," and W. I. Stanley and H. F. Hertel, "Statistics gathering and simulation for the Apollo real-time operating system," *IBM Systems Journal* **7**, No. 2, 74–102 (1968).
10. L. R. Huesmann and R. P. Goldberg, "Evaluating computer systems through simulation," *Computer Journal* **10**, No. 2 (August 1967).
11. I. D. J. Bross, "Models," *Management Systems*, John Wiley & Sons, New York, New York, 327–336 (1968).
12. D. M. Braddock and C. B. Dowling, *Simulation, Evaluation, and Analysis Language (SEAL)*, IBM Contributed Program Library, 360D 15.1.005, International Business Machines Corporation, Program Information Department, Hawthorne, New York. The reader should be aware that SEAL is not the only language that can be used to implement the model. Other languages such as SIMSCRIPT, CSS, and GPSS can also be used.