SIMULATION OF COMPUTER SYSTEMS USING AUTOMATICALLY GENERATED LOAD DESCRIPTIONS

Herbert D. Schwetman

Purdue University

ABSTRACT

A complete description of a computer system must include three groups of components:

1. The hardware components
2. The software components (the operating system), and
3. The load components (the stream of tasks to be processed).

When computer systems are simulated all of these components are found as features of the simulations. The hardware normally appears as a collection of tables and associated operational data. The operating system appears both as tables and as scheduling or assignment algorithms. The load is represented either as a sequence of requests for service (use of system resources) or as a set of functions which can generate such a sequence of requests. In many applications, the description of the load turns out to be the most difficult task.

This paper discusses some techniques and procedures which have been used to automatically generate the required load descriptions. The emphasis is on a request sequence generator which uses data found in a system event trace. One such technique, called trace-driven modeling, has been implemented at the Purdue University Computing Center. This implementation is discussed and its usefulness and accuracy are illustrated with actual data.

INTRODUCTION

The performance of a computer system depends on three major components:

1. The hardware system
2. The software system, and
3. The load.

Each of these components also affects the performance of a simulated computer system. Consequently, the designer of a discrete event system simulator must include each of these components in his simulator (1).

MacDougall (2) presents a clear and concise description of techniques which can be used to model the hardware and software components. The modeling of the load is a more difficult undertaking. This difficulty may be a significant factor in limiting the use of simulation models in the planning, design and operation of computer systems.

This paper briefly discusses discrete event simulation of computer systems, with emphasis on the load or task modeling phase of such simulation. A relatively new technique, called trace-driven modeling, is presented. The technique shows promise of being capable of accurately modeling the behavior of tasks executing in a computer system. A description of a trace-driven model of the CDC 6500 computer system in use at the Purdue University Computing Center is presented. Data which displays the degree of accuracy attainable with this model is included.

BACKGROUND

MacDougall has modeled a computer system as a collection of resources. A resource is a set of one or more elements with only one queue for waiting requests. Thus, in a system containing dual central processor units (CPU's) there would be a single CPU resource with two elements. On the other hand, in a system with several disk drives, each drive is a separate resource, because a task must request data from a particular drive. Hence there would be a separate queue for each drive.

In an actual system, the operating system has responsibility for initiating and terminating tasks and for responding to requests for service (access to resources) made by these tasks. In a multi-programming environment, there can be many concurrently executing tasks and thus many simultaneous requests for access to the resources of the system. The operating system coordinates these concurrent requests so that all accesses to the resources occur in a correct manner.

These task-control and access-coordination features are also present in a simulated system. In MacDougall's approach, much of the access coordination function can be modeled in the queue-handling mechanisms. For example, if requests which are waiting for access to a resource are enqueued in order of an ascending queue priority,

then several different resource scheduling disciplines can be implemented by merely manipulating the queue priority.

In summary, it is possible to model the resources which comprise a system as a collection of tables and queues. It is possible to model many aspects of the operating system as a collection of queue-handling algorithms, and both of these factors can be simulated using relatively straightforward techniques. Modeling the load imposed by a collection of tasks (equivalently the driving function for the simulator) is a more difficult problem.

One method of modeling the task load has been to use load-generating functions (3). In this method, whenever a request for service is needed by the simulator a function is invoked which produces the required values of the request. For example, if a point is reached at which a CPU request is needed, a CPU-burst-time-function is invoked to generate a time interval for the next CPU request. The degree of sophistication of this function can vary widely. The simplest function would return a constant value. A more complicated function would use characteristics of the modeled task and statistically derived parameters to produce a more realistic interval. The range of complexity possible is almost limitless.

Another method for modeling the load has been to prepare a set of job or task scripts (1, 4, 5). In this method, each active task has an associated list of system requests. Each request includes the name of the requested resource, the amount of the resource (often in units of time) and other pertinent information. The task load is modeled by having the simulator interpret these scripts as the sequence of resource demands. This second method can be viewed as a special type of the request-generating function described above; in this script method, each call to the generating function produces the next request from the script for that task.

Both of these methods have been used in documented system-simulation projects. The script method can produce realistic models of specified collections of tasks. In particular, a task script can be produced to recreate specified patterns of resource demands. The cost and difficulty in producing a set of task scripts can be significant, particularly if done manually. Task-description languages which decrease the required effort have been designed (4). Even with this type of aid, there is still a problem in gathering the information required to produce realistic scripts.

A statistically-derived request-generating function can be designed to generate request sequences which exhibit specified statistical properties. The required parameters can be automatically produced through analysis of system performance data if the system is available. The implementation of these functions can be much less difficult than the preparation of task scripts.

TRACE-DRIVEN MODELING

A relatively new technique for modeling the system load has appeared in the form of trace-driven modeling (6, 7). This approach uses task scripts similar to those described above, but with trace-driven modeling these scripts are automatically prepared from system event-trace data (8).

A system event-trace is a record of all significant events which occur during the operation of the system. One method of recording this data is to use a data-gathering routine integrated into the control section of the operating system (9). Events of interest, such as the request, assignment and release of system resources by tasks, can be identified, time stamped and recorded by the data-gathering routine. Subsequent analysis of this sequence of events can produce the task scripts suitable for use by a system simulator.

As an example of this script-generation process, consider the sequence of events depicted in Figure 1. It can be observed that there are two tasks (labeled A and B respectively) competing for three resources: the CPU, channel 1 and channel 2. The generated task scripts would be two sequences of requests for the resources with the required time of use (service time) associated with each request. Figure 2 displays the scripts which could be generated using the sequence found in Figure 1. It should be noted that all delays or waiting times have been left out of the scripts. Presumably, these would be functions of the system and the other competing tasks but would not be considered part of a task description.

FIGURE 1

Sample Event-Trace

Line No.	Time	Task	Event	Resource
1	.000	A	Asg	CPU
2	.002	A	Asg	CH1
3	.003	A	Rel	CPU
4	.003	B	Asg	CPU
5	.010	B	Asg	CH2
6	.011	B	Rel	CPU
7	.020	A	Asg	CPU
8	.021	A	Rel	CPU
9	.052	A	Rel	CH1
10	.052	A	Asg	CPU
11	.060	A	Rel	CPU
12	.068	B	Rel	CH2
13	.068	A	Asg	CH2
14	.068	B	Asg	CPU
15	.070	B	Rel	CPU

FIGURE 2

Automatically Generated

Task Scripts

Task	Resource	Amount
A	CPU	.002
	CH1	.050
	CPU	.010
	CH2	—
B	CPU	.007
	CH2	.058
	CPU	.003

A trace-driven model of the system shown in the example would have three simulated single-element resources corresponding to the CPU and channels 1 and 2 respectively. Associated with each resource would be a queue for requests waiting for access to that resource. The simulator would initiate each task (when appropriate) and then sequence through the task scripts, requesting the resources designated by the successive script entries for the indicated periods of time. The desired usage and queueing statistics would be accumulated during the period of simulated operation and the simulation would terminate after a preselected amount of time or when all of the task scripts had been processed. Figure 3 presents an event trace produced by the trace-driven model of the system.

FIGURE 3

Event-Trace of Simulated System

(with no CPU-Channel Overlap)

Line No.	Time	Task	Event	Resource
1	.000	A	Req	CPU
2	.000	A	Asg	CPU
3	.000	B	Req	CPU
4	.002	A	Rel	CPU
5	.002	B	Asg	CPU
6	.002	A	Req	CH1
7	.002	A	Asg	CH1
8	.009	B	Rel	CPU
9	.009	B	Req	CH2
10	.009	B	Asg	CH2
11	.052	A	Rel	CH1
12	.052	A	Req	CPU
13	.052	A	Asg	CPU
14	.062	A	Rel	CPU
15	.062	A	Req	CH2
16	.067	B	Rel	CH2
17	.067	A	Asg	CH2
18	.067	B	Req	CPU
19	.067	B	Asg	CPU
20	.070	B	Rel	CPU

At the Purdue University Computing Center an event-trace facility, a task-script generator and a discrete event system simulator similar to those described above have been implemented. The remainder of this paper describes this simulator and some experiments which have been conducted using these tools.

THE PURDUE TRACE-DRIVEN MODEL

The main computer at the Purdue University Computing Center is a CDC 6500 computer system. The operating system is the Purdue-MACE system, a locally derived version of the MACE-5 system available from CDC. This computer system and its several operating systems have been described in the literature (10, 11). The feature of the 6500/Purdue-MACE system which is of interest for this paper is the interface between user programs and the operating system, for it is demands for system service from user programs that will be the source of the task descriptions.

In the Purdue-MACE system, an active task occupies a control point (a virtual processor) and a contiguous block of main (central) memory. A task normally competes with other active tasks for use of one of the two central processing units. A task can request system service by presenting three-letter coded requests with parameters to the operating system. Some of these requests include:

1. Relinquishing use of the CPU resource until some event occurs,
2. Termination of a program,
3. I/O service, and
4. Use of other system facilities.

The I/O service and use of most system facilities are accomplished by the collection of eight peripheral processor units (PPU's) which are available to the system. The three-letter codes corresponding to the requests for service are treated as names of PPU programs. When these requests are presented to the system by a task, an idle PPU must be found and assigned to that task. The assigned PPU then locates the program in the system program library corresponding to the specified three-letter code and loads and executes that program. These programs normally execute for a short period of time (1-500 milliseconds); however, some system-related PPU programs may execute for much longer periods of time.

The Purdue-MACE system also includes an event-trace facility which can be invoked to capture and record events corresponding to assignments of resources in response to the demands by all of the active tasks. This facility is similar to the one described in Schwetman and Browne (9).

The initial version of the trace-driven model focused on the following resources: the peripheral processor units, the central processor units, and the central memory/control point resource. These resources were felt to be the most critical ones in the system. There is the obvious exclusion of the data channels and I/O devices from this list. The project proceeded on the assumption that these additional resources could be easily added after the feasibility of the proposed approach was determined.

The model consists of two major components: the task-script generator and the system simulator. The simulator is structured like the discrete-event system simulator described by MacDougall, with two changes: substitution of PPU's for I/O devices and the use of task scripts to generate the

sequence of requests for access to the resources of the simulated system.

After some initial experimentation with this first version, some modifications were suggested. The most obvious one was the installation of a round-robin service discipline for the CPU resource, with a slice-time of twenty milliseconds. This was necessary to achieve a reasonable degree of similarity in the behavior of the simulated system when compared to the real system.

Other changes were made which contributed to the realism of the model. One such change was directed at the system tasks being modeled. Such tasks consumed little or none of the CPU resource, but rather proceeded by making periodic calls for various PPU programs. The task scripts corresponding to these system tasks were modified to request no CPU service and to have an idle period corresponding to the elapsed real time inserted between successive requests for PPU service. This change produced a more realistic model of the behavior of these system-related tasks.

The task-script generator uses the chronologically ordered system event-trace to build a task script for each task (job) encountered in the data. The scripts were stored on a direct-access file along with an index so that the simulator could access the scripts in a completely arbitrary order. The task in-core time, the initial task memory requirement and the initial task priority were also recorded for use by the task-scheduling section of the model.

A typical script for a task begins with an entry which requests the initial priority and memory for the task. This is followed by a series of requests which alternates between a request for CPU service and a request for PPU service. Each request includes a time value which is the length of the requested service. This alternating series continues until the end of the script is reached. Some of the PPU requests can proceed in parallel with CPU activity while others must be the sole activity associated with a task. These differences are denoted by the name of the PPU program and are properly handled by the simulator.

EVALUATION OF PURDUE TRACE-DRIVEN MODEL

System simulators can be evaluated according to several criteria. One criterion could be ease-of-use. The simulator just described is easy to use provided that the structure of the system and/or the scheduling algorithms are not altered. The numbers of elements within a resource, the degree of multiprogramming, and the length of the CPU time-slice are all parameters for the simulator. The task description activity is completely automated and presents no difficulty to prospective users of the simulator.

Another criterion is the amount of computational resources required to perform the simulation. Table 1 presents data which provides a basis for the claim that these demands are not excessive.

TABLE 1

Typical Computing Resources Required by Purdue Trace-Driven Model

	Run 1	Run 2
Number of Resource Assignments	45360	50473
Elapsed Simulated Time (seconds)	581.5	658.3
CPU Time Used (seconds)	18.3	23.3
Elapsed Real Time (seconds)	55	67

Still another criterion that must be met is that the simulator be capable of providing the necessary information to the user. This is a matter which is dependent on each instance of use of the simulator and will not be discussed further. The simulator is capable of providing all of the standard usage and queueing statistics typically found in system simulators.

An important remaining criterion concerns the accuracy of the simulator. Measures, such as total CPU time, average CPU burst time, and PPU times do not describe the accuracy of the simulated system, since these values are all directly available in the task scripts. However, for the sake of completeness, these are summarized in Table 2. Other measures, such as the in-core times for tasks and the time required to complete a set of tasks are meaningful measures of accuracy and will be presented here to support claims of good accuracy. In particular the differences between the observed and the simulated in-core times for each of the simulated tasks are tabulated and used to assist in determining the accuracy of the technique.

TABLE 2

Comparison of Actual and Simulated System Performance Data

	Monoprogrammed		Multiprogrammed	
Number of Jobs	20		20	
Number of Systems Tasks	3		3	
	Actual	Simulated	Actual	Simulated
Elapsed Time (seconds)	658.329	658.290	388.658	388.667
Total CPU Time (seconds)	496.031	472.698	501.279	481.415
Number of CPU Assignments	17126	31588	25994	32151
Average CPU Burst (seconds)	.029	.014	.019	.015
Total PPU Time (seconds)	488.061	487.641	443.909	443.909
Number of PPU Assignments	18862	18862	13186	13186
Average PPU Burst (seconds)	.026	.026	.034	.034

Table 3 presents the results of a test-run consisting of twenty benchmark jobs executed in a monoprogrammed environment. The results of this test suggest that trace-driven modeling can accurately simulate the behavior of programs in a monoprogrammed environment. Table 4 summarizes the relative errors for the simulated in-core times for the same twenty jobs executed in a multiprogrammed environment. In the simulated multiprogrammed environment it is much more difficult to control the conditions in which each task is executed. In particular, the simulated and actual systems do not use the same scheduling strategy for

selecting the tasks for occupancy in memory. For example, some tasks executed in situations in which the number of simultaneously executing tasks in the simulated system was not the same as in the actual system. In an attempt to fairly summarize the error data, the simulator was run using three different task scheduling strategies and the lowest value for the relative error for each task is presented as the last column of data in Table 4.

TABLE 3

Summary of Data

Simulated In-Core Times for Monoprogrammed Environment

Job	Name	In-Core Time Actual	In-Core Time Simulated	Relative Error
1	MW75715	9.615 sec.	8.410 sec.	12.5%
2	MH31811	3.387	3.523	-4.0
3	MH75816	66.999	76.735	-14.5
4	MR90218	55.232	47.276	14.4
5	MS47419	41.763	36.464	12.7
6	MM86424	33.603	31.178	7.2
7	MB74426	5.983	5.955	.5
8	MD52829	2.144	2.149	-.2
9	MD52930	2.058	2.051	.3
10	MS47031	43.045	39.176	9.0
11	ME21619	20.766	20.596	.8
12	MJ59624	1.516	1.467	3.2
13	MB94429	73.328	77.532	-5.7
14	MH20428	34.562	36.234	-4.8
15	MU06122	20.710	20.208	2.4
16	MB11933	2.017	2.038	-1.0
17	MU58229	2.864	3.058	-6.8
18	MK30051	20.051	19.294	3.8
19	MW82792	63.272	63.324	-.1
20	MY25417	111.069	110.805	.2
TOTALS		613.984 sec.	607.473 sec.	1.1%

Average of Absolute Value of Relative Errors	5.2%
Maximum Relative Error	-14.5%
Minimum Relative Error	.1%

TABLE 4

Summary of Relative Errors

Simulated In-Core Times for Multiprogrammed Environment

Job	Name	Actual In-Core Time	Relative Errors: First Come First Served	Largest Priority First	Smallest Priority First	Best
1	MW75715	13.769 sec.	9.9%	9.2%	9.9%	9.2%
2	MH31811	8.242	-.4	-1.2	-.4	-.4
3	MH75816	65.537	-8.2	-8.2	-8.2	-8.2
4	MR90218	60.234	16.7	12.7	14.2	12.7
5	MS47419	44.970	16.3	16.3	16.3	16.3
6	MM86424	38.507	-39.3	-77.9	-31.3	-31.3
7	MB74426	9.215	.7	.1	-3.6	.1
8	MD52829	2.250	1.6	-6.3	.2	.2
9	MD52930	1.999	-1.7	-10.7	-4.7	-1.7
10	MS47031	43.807	-68.3	2.9	-44.1	2.9
11	ME21619	21.682	-37.4	-41.9	-50.0	-37.4
12	MJ59624	1.346	-15.5	3.9	-3.7	-3.7
13	MB94429	111.173	11.2	-26.6	24.8	11.2
14	MH20428	79.315	-14.7	-27.4	-5.7	-5.7
15	MU06122	22.987	-87.9	-79.3	-37.7	-37.7
16	MB11933	2.318	5.5	-21.3	3.7	3.7
17	MU58229	9.386	-2.5	-2.8	-2.5	-2.5
18	MK30051	24.486	10.9	12.5	6.9	6.9
19	MW82792	100.630	-16.8	-28.6	1.9	1.9
20	MY25417	134.708	-29.9	-34.9	-18.9	-18.9
TOTAL		796.561				

Average of Absolute Values of Best Relative Errors	10.6%
Maximum Best Relative Error	-37.7%
Minimum Best Relative Error	.1%

The results of these tests show that the current model is not capable of exactly reproducing task behavior. However, the tests do show that the trace-driven model can provide a level of accuracy sufficient to permit the model to be used as an aid to system design.

SUMMARY

Some simulation projects require accurate modeling of the behavior of specified tasks which are executed by the simulated system. Trace-driven modeling is a technique which should be capable of achieving such accuracy. In order to test this assertion, a trace-driven model of the CDC 6500 in use at the Purdue University Computing Center was implemented. In addition to demonstrating the ease-of-use which can be achieved by the automatic task-script generator, the test also produced data which permitted the accuracy of the model to be determined. The data does suggest that the in-core times for jobs executing in a monoprogrammed environment can be accurately modeled. The results for the multiprogrammed environment were less conclusive.

The technique is not applicable in all situations in which computer systems are simulated. For example, the job stream and/or the system which are to be modeled may not be available. Also, many systems are not equipped with a system event-trace facility from which suitable task scripts can be generated.

The model implemented at Purdue is not complete. There are several items which can be modified or added which should improve the accuracy of the model. The goal of this effort has been the production of a tool which will allow system designers to "try out" features using the simulator before such features are actually implemented. It appears that automatically generated system loads as described in this paper offer the two advantages of ease-of-use and accuracy and will be used in subsequent system simulation projects.

REFERENCES

1. Seaman, P. H. and R. C. Soucy, "Simulating Operating Systems", IBM System Journal (8, 4), 1969, pp. 264-279.

2. MacDougall, M. H., "Computer System Simulation: An Introduction", Computing Surveys (2, 3), September, 1970, pp. 191-242.

3. Katonak, P. R., "Use of Performance Analysis Statistics in Computer System Simulation", Proceedings Winter Simulation Conference, 1971, pp. 317-326.

4. Stanley, W. I. and H. F. Hertel, "Statistics Gathering and Simulation for the Apollo Real-Time Operating System", IBM Systems Journal (7, 2), 1968, pp. 85-102.

5. Neilsen, N. R., "An Analysis of Some Time-Sharing Techniques", Communications of the ACM (14, 2), February, 1971, pp. 79-90.

6. Cheng, P. S., "Trace-Driven System Modeling", IBM Systems Journal (8, 4), 1968, pp. 280-289.

7. Sherman, S., Baskett, F. and J. C. Browne, "Trace-Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System", Communications of the ACM (15, 12), December, 1972, pp. 1063-1069.

8. Drummond, M. E. Jr., Evaluation and Measurement Techniques for Digital Computer Systems, Prentice-Hall, Inc., 1973, pp. 122-188.

9. Schwetman, H. D. and J. C. Browne, "An Experimental Study of Computer System Performance", Proceedings 25th ACM National Conference, 1972, pp. 693-703.

10. Abell, V. A., S. Rosen, and R. E. Wagner, "Scheduling in a General Purpose Operating System", Proceedings FJCC, 1970, pp. 89-96.

11. Thornton, J. E., Design of a Computer System: The Control Data 6600, Scott-Foresman and Company, 1970.