SOME EXPERIMENTS IN SIMULATING OS/360 FROM PERFORMANCE DATA

Leonard Bass,[1] Thomas Santos,[2]

and Michael Sheets[3]

[1]University of Rhode Island

[2]Department of the Navy

[3]IBM Corporation

ABSTRACT

Recent work has indicated that the time a program spends in memory is the primary determinant of the turnaround of the program. We constructed a simulation model of our OS/360 system, using as input performance data gathered by a software monitor, and experimented with differing models of the operating system, program behavior, and hardware configurations to determine an appropriate trade-off between the complexity of models used and the accuracy of the simulation. We constructed a fairly simple model which simulated, from resource requirements of programs, the time a collection of programs resided in memory to within 10% accuracy for the actual completion time of the various programs.

The data gathered gave, between any two sample points, the central processor time spent processing on interrupts, on various system tasks, on various problem program tasks and on the large time-sharing task which runs in our system. It also gave the number and total duration of the input-output requests from each task to each device. This data was sampled every time a program was introduced into memory or terminated and left memory.

The simulators we constructed had components for the operating system (in various levels of detail), problem program behavior (we tested various distributions of processor burst time and input-output time) and also a component for the input-output configuration (we tested various models of channels and devices with associated queueing). The test of the accuracy of the simulation was how well it compared with the times taken by the actual programs. The results yielded insight into the nature of program behavior as well as how programs interact with the operating system.

INTRODUCTION

Simulation plays an important role in analyzing the performance of computer systems, both existing and projected [5]. Recent work in simulating a computer system from accounting data has shown that the most critical factor in the performance of a system is the amount of time programs spend in main memory of the computer [7]. We here present the results of a series of simulations from performance data which indicates that both an appropriate model of the operating system and an appropriate model of problem programs are important factors in simulating the residence time of a program in main memory. In the process, we construct a deterministic simulation model which is accurate to about 10% in simulating the turnaround time of a collection of programs. We also develop a model of the operating system which should be useful as a basis for collecting performance data.

In any simulation of a computer system, there are four components [6]. The resource requirements of the problem programs have to be specified. Also, the hardware, the operating system, and the behavior of the program has to be modeled.

RESOURCE REQUIREMENTS

We specified the resource requirements from performance data which was collected in the actual operating environment over an extended period of time. We will briefly describe the type of data that was collected and the installation at which it was collected.

The University of Rhode Island, at the time this data was collected, had an IBM 360 Model 50 with half a million bytes of two microsecond core and one million bytes of four microsecond core. OS/MFT-II with HASP was the primary operating system and CALL-OS handled a timesharing system which appeared as a problem program to the operating system.

TABLE 1

Period Information

Period	Uncorrected: In Seconds	Uncorrected: Number of Intervals	Uncorrected: Error	Intervals Violating Assumptions Removed: In Seconds	Intervals Violating Assumptions Removed: Number of Intervals	Intervals Violating Assumptions Removed: Error	Utilization	I/O Request per minute Wall Time
1	780	77	31.05%	502	59	12.62%	82%	1590
2	3044	124	24.53%	2139	108	5.81%	89%	950
3	1142	106	8.74%	1141	105	8.73%	93%	1240
4	681	18	25.06%	470	17	5.77%	47%	1700
5	6106	119	3.79%	5784	113	.94%	99%	540

A software monitor was inserted into the system and this monitor yielded, between any two sample intervals, the time the processor spent processing on any identifiable task, including system tasks. It also yielded the number and duration of the input-output requests issued by each identifiable task, again including systems tasks, to each particular input-output device. This information was sampled at the beginning and end of any job step in the interval. Figure 1 gives an example of how these job steps interact. The intervals for which we can recover resource usage are t_1-t_2, t_2-t_3, etc. These intervals vary in length, but for each interval we can account for the totality of processor time as well as the number and duration of input-output requests from any task to any device.

From a vast amount of data collected during the Spring of 1972, five periods were chosen to reflect the differing rates of input-output activity that our workload undergoes. These periods range in length from 11 minutes, 21 seconds to one hour, 36 minutes and 24 seconds. The processor utilization ranges from 99% to 47%, and the input-output rate ranges from 540 input-output requests per minute of wall time to 1700 per minute of wall time. Table 1 lists some relevant facts for each period.

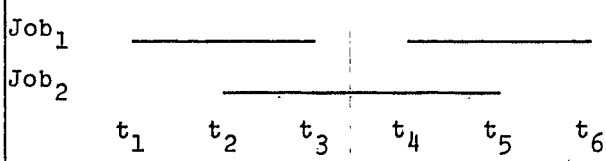
BASIC MODEL

Since we were interested in simulating the residence period of a program, we did not model any job queueing or scheduling. We used the programs that were scheduled by the operating system as they occurred.

The measurement of actual input-output times freed us initially from simulating any channel activity. This allowed us to examine models of the operating system and the problem program independently of most hardware considerations.

FIGURE 1

Data Collection



Between any two sample points, for example t_2 and t_3, we know all of the time the processor spent in any state and we know how many requests job 1 made to device 1, how many to device 2, etc. We also know how long these requests took to service, per device.

For example, between t_2 and t_3, the processor spent 3 seconds processing on job 1, 2 seconds processing on job 2, .5 seconds processing interrupts, and 6 seconds idle. Also job 1 made three requests to device 1 and the service time for these requests totaled 2 seconds. Job 2 made 5 requests to device 1 and the service time for these totaled 7 seconds.

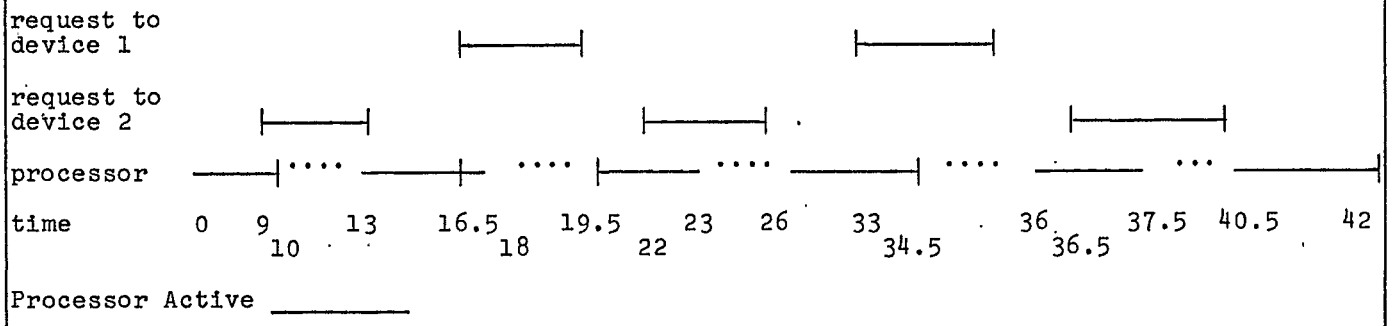
We initially modeled individual problem programs by defining a processor burst for each problem program with respect to each input-output device. This burst is defined by dividing the total amount of processor time used by that program by

FIGURE 2

Basic Program Model Example

Program A between times t_1 and t_2 used 30 seconds of processor time. It made 2 requests to device 1, totaling 6 seconds and 3 requests to device 2 totaling 12 seconds. We assume 10% overlap.

The processor quantum of Program A with respect to device 1 is 15 seconds. Each request to device 1 takes 3 seconds. The first request from Program A to device 1 is issued after 13.5 seconds of processor time have elapsed. 1.5 seconds of this processor quantum is overlapped with the input-output request. The processor quantum with respect to device 2 is 10 seconds. Each request to device 2 takes 4 seconds and the first request to device 2 is issued after 9 seconds have elapsed.



the number of input-output requests to that device by that program. This yields the amount of processor time accumulated between requests to that device. The duration of each request is the total time doing input-output to that device divided by the number of requests to that device. The actual request is considered to be made after a fixed percentage of the burst is completed. The processor then completes its quantum, overlapping the rest of the quantum with the input-output. The program is then blocked until completion of the input-output request. The percentage of the quantum that is overlapped with the input-output is called the overlap percentage. See Figure 2 for an example of this model.

The activity that we modeled fell into two categories. First, some activities were foreground activities. They did not occur as sequences of job steps, but consumed resources sporadically throughout an interval. The timesharing system and HASP fall into this category. The other activities were those of problem programs. A problem program does proceed as a sequence of job steps and when not in a queue, is always either processing or doing a data transfer. During the periods that we examined, at least one problem program was always active, along with varying levels of activity of the foreground programs.

Two assumptions about problem programs need to be clearly stated at this point. First, we assumed the program becomes blocked only if input-output is actually taking place. In certain intervals this does not happen, e.g. operator intervention is required. The most troublesome of these intervals were detected by comparing the sum of all input-output times of all the programs to the time the processor spent in the waiting state. If the wait state time is greater then at least one of the programs must have become blocked when not doing input-output. These intervals are considered separately.

Secondly, we assumed in our model that each program had at most one request pending to a particular device at any point in time. This assumption was also occasionally violated, detected by comparison of the total input-output time to each device to the length of the interval and treated separately. Many of the apparent violations of this assumption occurred because of our manner of timing tape rewinds and were not actually violations.

We collected all system activity (both processor and input-output) into an independent activity, and treated it the same as a foreground program. These were divided into compute burst quanta just as were the problem programs.

The other aspect of the basic model is the dispatching priority. Highest dispatch priority was given to the system activities, next highest to HASP, next to the timesharing system, and the remaining three problem programs were dynamically

dispatched using an option of HASP. This dynamic dispatching has been the subject of much independent study [9,8,1].

CRITERION FOR MEASURING SIMULATIONS

Our measure for the accuracy of the simulation is a comparison of the length of the simulated intervals to the actual length of the intervals. We used a measure of the turnaround error in the simulation calculated as the sum of the absolute value of the differences over each interval expressed as a percentage. Formally, if x_i represent the actual interval length and y_i represent the simulated interval length, we have:

$$\text{turnaround error} = \frac{\Sigma|y_i - x_i|}{\Sigma x_i}$$

This is similar to one of the measures used by Beilner and Waldbaum [3].

RESULTS FROM THE BASIC MODEL

The main result from our simulation of these five periods with the basic model is that a tremendous amount of the error in the simulation is attributable to the two assumptions mentioned above. Removing the intervals flagged as violating these assumptions, reduced the error by at least a factor of two in every period, save one where only one interval of one second duration was flagged. In period 4, by removing one interval where operator intervention was required, we reduced the error by almost a factor of five.

These figures demonstrate the importance of these two assumptions. Any simulation from performance data must make allowances for the occurrence of operator intervention. Also, the possibility of multiple requests pending simultaneously from one program to one device must be allowed for either in the collection of the data or the simulation model or both.

For the rest of this paper, we shall present our results for the intervals that remain after the intervals that violated the assumptions were removed. Table 1 gives the pertinent data about these periods.

MODELS OF OPERATING SYSTEMS

Implicit in the data collection was a particular model of the operating system. This was described above and assumes that the only aspect of the operating system necessary to be modeled was the dispatching priority. Dispatching priority is in fact important, but modeling other aspects of the operating system are equally as important.

The most significant aspect of the operating system models investigated lies in the modeling of the input-output activity attributed to system overhead. This activity falls into two categories. Some of it does not block the problem program (e.g. operator interaction with the system) and the rest of the system input-output activity does block the problem program (e.g. loading a transient system function). Since the data that was collected did not contain information allowing the system input-output activity to be distributed to the appropriate problem program, this time was allocated to the problem program that finished first within the interval. In no case did this increase the turnaround error by more than 1 1/2% (period 1), and in the case of period 4, the turnaround error dropped from 26.12% to 5.77%. This provides evidence that the system model input-output chosen does have a large impact on the results.

A similar experiment was made with respect to interrupt processing times. These times were prorated among the programs according to the number of input-output requests issued. This resulted in a small decrease in turnaround error of about 1%.

Dispatching priority was also studied in two ways. The dynamic priority allocation of HASP was turned off which caused an increase in turnaround error of about 1 1/2% with one period (period 1) showing a 7% increase.

The timesharing system operates at two different dispatching priorities. A percentage of its time (depending on the load [4]) is spent in lowest dispatching priority. This was modeled by having half the processor time accumulated by the timesharing system dispatched at lowest priority. This caused up to 4 1/2% variation in turnaround error (one period increasing and one decreasing). Since, again, the data did not include information about the actual dispatching, our only interpretation of this is that the modeling of dispatching priority does have a significant effect on the accuracy of the simulation.

These results have implications for both performance data collection and simulations based on performance data. The model of the operating system which is used does have a significant effect on the accuracy of the simulation. The data which is collected must include allocation of system activities to the appropriate problem program and it should include enough detail to model dispatching prior-

ity. Table 2 gives the specifics of the various operating system models.

TABLE 2

Models of Operating System

Model A is the model of the operating system used in the remainder of the study. It attempts to correct for system activity (both processor and input-output). It models the dynamic dispatching algorithm of HASP and the low priority utilization of the timesharing system.

Model B differs from Model A only in its attempt to correct for system input-output activities.

Model C differs from Model A in its attempt to correct for system processor activity.

Model D differs from Model A by not modeling the HASP dynamic dispatching algorithm.

Model E differs from Model A by not modeling the partial low dispatching priority of the timesharing system.

Model 1

Period	A	B	C	D	E
1	12.62	11.47	15.63	19.70	15.75
2	5.81	7.64	6.50	7.18	5.60
3	8.73	8.25	9.60	10.18	9.24
4	5.77	26.12	6.93	8.21	9.99
5	.94	.80	1.09	.82	1.17

(all numbers are percentages)

PROBLEM PROGRAM MODELS

Using a fixed model for the operating system (Model A of Table 2) we next investigated the effect of varying the model of the problem program that was used. Two components of the problem models were investigated. One was the effect of varying the percentage of each processor burst that was overlapped and the other effect of using different distributions of processor burst time and input-output request time.

Of these two components, the percentage of each processor burst overlapped had a significant effect on the observed error and the distribution of the burst and request times has virtually no effect on the observed error.

Five different overlap percentages were chosen. The results of these different percentages are given in Table 3. Independent examination of this same data has shown that for our system the appropriate overlap is 20% [2], but the significant point is the effect of any overlap at all. The turnaround error with 12% overlap is generally less than 2/3 of the error with no overlap grows to 24% and increases again as it grows to 100%.

TABLE 3

Effect of Varying Overlap Percentage

Percentage

Period	0	12	18	24	100
1	21.15	12.62	11.08	9.43	9.72
2	7.67	5.81	5.95	6.07	8.32
3	14.20	8.73	8.09	7.04	5.75
4	9.64	5.77	4.59	4.74	9.86
5	1.60	.94	.88	.81	.31

(all numbers are percentages)

Some nine different combinations of processor burst distributions and input-output request time distributions were examined. We assumed that the lengths of the processor bursts were constant or that they were distributed uniformly, according to a normal distribution, or according to a gamma distribution. We also made assumptions about the distribution of the input-output request times. The largest variation between any two of the models studied caused less than 2.6% difference in turnaround error. The model that performed close to the best was the basic model, described above. Thus, in simulating the behavior of problem programs, it increases the error to use more elaborate distributions and it also adds additional overhead.

DEVICE AND CHANNEL MODELS

Up to this point, the various models we have examined have used measured values for all resource usage and thus, our results on models of operating systems and program behavior should pertain to other installations using OS/360. Our investigation of differing input-output models, however, is influenced heavily by the particular configuration and job stream used. For this reason we will be very brief in describing our results.

We examined three different queueing models using as the input-output times a fixed set of times plus the times spent in queues. These replaced the measured input-output times as input for each interval. The three models were: 1) no queueing,

2) device queueing only, and 3) channel and device queueing. The errors from these three models were virtually indistinguishable. The errors were sometimes higher than those from the basic model and sometimes lower, apparently unaffected by input-output rate or processor percentage. The interesting aspect of this study was that for several periods the errors using the queueing models were substantially higher than the error of the basic model. This reflects the difficulties incurred when applying global input-output times to particular cases. No matter which times are chosen, for some intervals they are totally erroneous.

CONCLUSION

We conclude that it is feasible to simulate the main memory residence time of programs in a real computer system within a fairly small error.

It is important when doing this simulation, however, to correctly attribute the various system activities to the problem programs that request these activities. It is not necessary to attempt elaborate distribution of problem program input-output times, but it is important to allow for some overlap of individual programs input-output and processing.

Finally, when simulating from performance data, it is critical that periods during which human intervention is required be identifiable.

REFERENCES

1. Bass, L. J. "On Optimal Processor Scheduling for Multiprogramming". SIAM Journal on Computing, Vol. 2, No. 4 (December 1973).
2. Bass, L. J. and Smith, E. "Some Aspects of Program Behavior in OS/360", Computer Science and Experimental Statistics Technical Report 73-109, University of Rhode Island, April 1973.
3. Beilner, H. and Waldbaum, G. "Statistical Methods for Calibrating a Trace Driven Simulator of a Batch Computer System", Statistical Computer Performance Evaluation, Academic Press, New York, 1972, pp. 423-459.
4. IBM, "CALL-OS Executive and Utility System Manual", form number GY20-0529-0, IBM Corporation, White Plains, New York.
5. Lucas, H. C. "Performance Evaluation and Monitoring", Computing Surveys, Volume 3, No. 3 (September 1971), pp. 79-91.
6. MacDougall, M. H. "Computer System Simulation, an Introduction", Computing Surveys, Volume 2, No. 3 (September 1970), pp. 191-204.
7. Noe, J. D. and Nutt, G. J. "Validation of a Trace Driven CDC6400 Simulation", Proc. SJCC, 1972, Volume 40, AFIPS Press, Montvale, New Jersey, pp. 749-758.
8. Sherman, S.; Baskett, F.; and Browne, J. C. "Trace Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System", CACM 15,12 (December 1972), pp. 1063-1069.
9. Stevens, D. F., "On Overcoming High-Priority Paralysis in Multiprogramming Systems: A Case History", CACM 11,8 (August 1968), pp. 539-541.