

A STUDY OF RESPONSE TIMES  
UNDER VARIOUS DEADLOCK  
ALGORITHMS AND JOB SCHEDULERS

Stephen W. Sherman, Institute  
for Computer Applications in  
Science and Engineering;  
John H. Howard, Jr., and  
James C. Browne, The University  
of Texas at Austin

Operating systems, deadlocks, scheduling, trace-driven modeling, response times.

Abstract

A trace-driven model is used to study the effects of various schedulers and deadlock control algorithms, and their interactions, on response times in a general-purpose operating system. Jobs' requests for memory and processors are extracted from a production load and used to drive a detailed simulation program. The simulation results show that response time is more sensitive than CPU utilization to differences between schedulers and deadlock control algorithms. Preemptive scheduling improves response time but degrades CPU utilization. Preemptive deadlock control algorithms improve both measures of performance. There are significant interactions between schedulers and deadlock control algorithms. Deadlock control algorithms can not be expected to optimize resource utilization.

I. Introduction

A variety of algorithms for dealing with the deadlock problem in operating systems have been proposed and compared qualitatively.<sup>1,2,3,4</sup> This paper reports a quantitative study of the effects of deadlock control algorithms and job schedulers on two significant measures of system performance: interactive response time and CPU utilization.

Trace-driven modeling<sup>5,6,7</sup> is the vehicle used for this study. It is a simulation technique based on a detailed job load extracted from a production system, and has been used to study other system algorithms such as CPU scheduling.<sup>5,7</sup> A previous study of deadlock control algorithms, considering only the effect on CPU utilization, appears in <sup>8</sup>. The more sensitive measure of response time for interactive jobs is considered here. Response time is particularly significant in the study of deadlock control since it is in interactive systems that competition for resources is most intense and transactions occur at high rates. The system model used here is more complete than that used for the previous preliminary report.

II. Trace-Driven Modeling

Trace-driven modeling is a simulation technique which uses actual job activity streams extracted from an operating system running under a production load. An event recorder, embedded in the system software, intercepts and records events

concerning job activity on an external medium, such as a magnetic tape. Each recorded event identifies 1) a job, 2) a resource and the quantity used, 3) the time the event occurred, and 4) the type (request, assignment, preemption, or release) of transaction performed.

The event tape is postprocessed to separate the jobs' event streams. The effects of competition between jobs are removed. These effects usually take the form of delays between requests or preemptions and the subsequent (re-)assignments of resources. The result of the postprocessing is a detailed sequence (trace) of the resource demands of each job observed. The traces are used to drive an otherwise conventional but very detailed simulation model. Figure 1 summarizes the main steps in trace-driven modeling.

There are two advantages in trace-driven modeling. A detailed job load extracted from an event trace is both realistic and reproducible. Since the simulator can be set up to model the system from which the event tape was produced, it is possible to validate the load and simulator by comparing the simulator's performance measures with those taken directly from the original event tape.

III. System Characterization

Trace data for this study is taken from a CDC 6600<sup>9,10</sup> under the control of a locally-written multiprogramming system called UT-2.<sup>5,11,12</sup> Jobs are swapped between central memory and extended core storage (ECS), with disks used in case of ECS overflow and for file storage. Central memory allocation is dynamic, using a base and bound register for each job. The maximum degree of multiprogramming under UT-2 is 13.

A job may use the CPU and perform input/output only when it is in central memory. Input/output requires the use of a peripheral processor. The peripheral processors also perform most system functions, including swapping and controlling devices, such as multiplexors, which need continuous attention, and they serve as intelligent channels. One peripheral processor, called the monitor, is devoted to basic message switching and resource allocation. (Certain system functions are performed by a central processor monitor.) The software event recorder is embedded in the peripheral processor monitor. Five of the peripheral processors are available for user input/output and swapping.

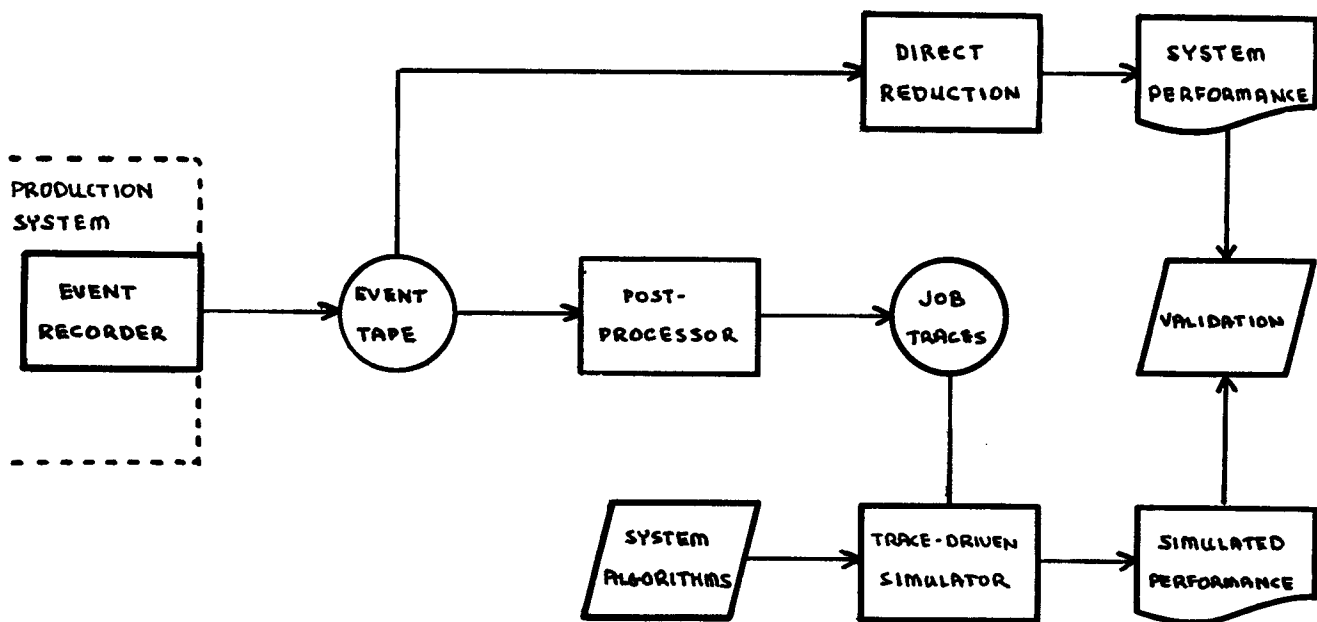


Figure 1. Trace-Driven Modeling

#### IV. Load Characterization

The UT-2 system is a suitable data source for this study because it is a general purpose system which supports a mixed batch and interactive job load and because it contains the necessary software event recorder. The data used here comes from the end of a semester at UT-Austin, on December 13, 1972. This was a period of heavy load. The system, during daylight hours, supported 35 to 45 interactive users logged in simultaneously, and consistently ran 5000 to 6000 batch jobs during a 22-hour day.

Measurements were taken over two periods, a relatively long one of 30 minutes and 1400 interactions and a shorter one of 3 minutes and 220 interactions. Interactive load is typically cyclic, with a peak in the afternoon. The long measurement period was in the morning, with from 30 to 35 users logged in, and shows the system behaving well. The short period was in the afternoon, with 40 to 45 users, and exhibits poor performance due to overload. It was difficult to obtain useful data over a longer period in the afternoon due to data being lost by the event recorder.

#### V. System Model

Figure 2 shows the model of job processing used here. When a user completes input from his terminal, his job leaves the think state and enters a queue for central memory. The scheduler eventually selects the job and initiates a swap-in. The job then cycles between CPU and input/output usage until processing is complete or the scheduler decides to preempt it, at which time the job is swapped out and returns to the think state or rejoins the memory queue.

The three nodes labeled "swap in", "I/O" and "swap out" each require the use of a peripheral processor. The model deals with contention for peripheral processors but does not otherwise distinguish I/O requests.

Response time usually is defined to be the interval from the moment a job leaves the think state until the moment it completes all processing necessary for the interaction. This study uses a different quantity, sometimes called "reaction time", which starts when the job leaves think state and ends when the swap in is completed. Further processing, including preemptions by the job scheduler (as a result of which a job reenters the memory queue, bypassing the think state) is ignored. This approach avoids the variations in response time introduced by varying compute requirements of interactive jobs and by the presence of batch jobs, for which the conventional response time is identical with the jobs' entire processing time. There is little difference in the case of truly interactive jobs between the two measures. A typical reaction time for the UT-2 system is 300 to 400 milliseconds, followed by a compute time, for jobs using text editors, BASIC, and similar processors, as short as 6 milliseconds.

#### VI. Schedulers

The job scheduler in UT-2 assigns to each job a cost equal to the product of the job's current memory requirement and the amount of service (CPU+I/O) it needs before completing its current transaction. For batch jobs the service requirement is simply the job's time limit less the time spent so far. An estimate, based on the time spent so far, is used for interactive jobs. The scheduler sorts the jobs into order of increasing cost, and then scans the sorted list (least-cost first) selecting any job which will fit into the memory left over from previously-selected jobs. To prevent interactive jobs (which usually have small costs) from monopolizing the system, the scheduler selects at most four of them. Finally, it compares its new selections with the set of jobs currently in memory and generates swap in and out requests to reconcile the differences. Swaps out may reflect either completion of processing or preemption of central memory.

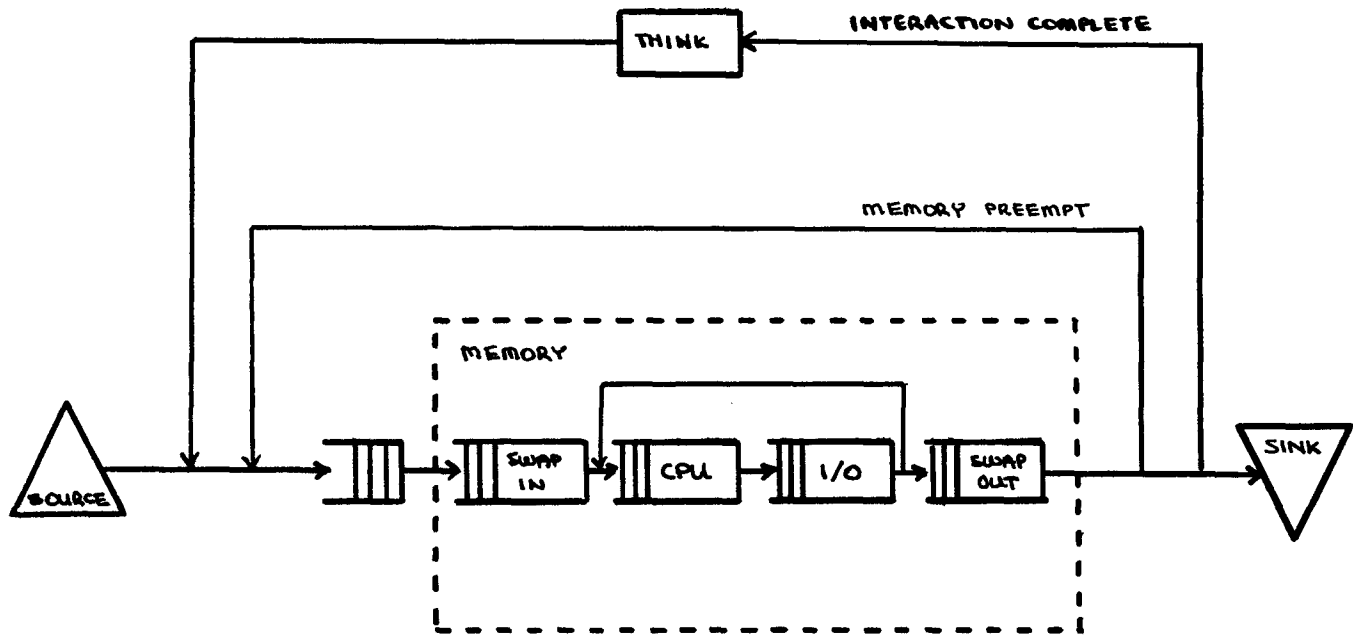


Figure 2. System Model

The standard UT-2 scheduler, referred to as S4, runs only when all of the swaps it requested in its most recent previous run are completed. This is unsatisfactory in combination with certain of the deadlock control algorithms, for the scheduler could become involved in a deadlock if one of the swapping peripheral processors is blocked. Consequently, S4 is used for validation only. A new scheduler, S1, is derived from S4 by not waiting for swaps in to complete (swaps out cause no difficulty) and is used as one of the schedulers to be compared.

The second scheduler evaluated, S2, differs from S1 only in that it is non-preemptive (S1 preempts jobs whenever cheaper jobs arrive in the memory queue). S2 does not allow one batch job to preempt another, although it does allow preemption by interactive jobs. The third scheduler, S3, attempts to run the 13 cheapest jobs regardless of the space constraint. The purpose of S3 is to evaluate the performance of deadlock control algorithms under overload conditions and without the help of a memory scheduler.

The CPU scheduler uses an 8-millisecond round robin. Previous studies<sup>7</sup> have established the effectiveness of this discipline, so it is not varied in this study. The round robin insures that all eligible jobs will get CPU service, so the CPU cannot enter into deadlocks and is not considered by the various deadlock control algorithms.

The remaining resource, peripheral processors, is scheduled on a first-come, first-served basis with no preemption, except as modified by the deadlock control algorithms.

#### VII. Deadlock Control Algorithms

Four algorithms for dealing with deadlocks are studied. D1, immediate preemption, is the technique actually used in the UT-2 system. If a job's request for memory cannot be satisfied immediately,

the job is swapped out and rejoins the memory queue. Peripheral processors (with the exception of the one through which the job requests memory) are not preempted, for they reliably complete operation without waiting for other resources. A swap out is delayed until the job's current peripheral processor activity ends. A job which is to be swapped out cannot use the CPU nor call for additional peripheral processors.

D2, complete assignment, prevents deadlocks by assigning to a job at initiation all of the resources it will ever need. This is a conservative and non-preemptive technique. It requires knowledge of upper bounds for job resource needs.

D3 is a standard detection and recovery scheme. Whenever a job's request for additional resources cannot be satisfied, a detection algorithm as described by Shoshani and Coffman<sup>3</sup> is run. If a deadlock is detected, the resources currently held by the requesting job are preempted.

D4 is the "banker's algorithm" for avoiding deadlocks as described by Habermann.<sup>4</sup> Resource assignments are made only when the system can find at least one safe sequence in which it can run all jobs currently holding resources (including the prospective assignment). Like D2, the avoidance algorithm is non-preemptive and requires advance knowledge of jobs' resource requirements. This knowledge is derived by looking ahead in the job event streams. Thus it is the best possible estimate. Poorer estimates would degrade the performance of systems using D2 or D4.

#### VIII. Validation

The UT-2 system runs with an input queue of 200 to 400 jobs, including the interactive jobs. About 100 of these jobs will run, and thus produce event streams, in a half-hour measurement period. Therefore the load presented to the simulated scheduler may be distorted. If the simulation is

started with all observed jobs present, and no new arrivals occur, the scheduler will run the smallest jobs first and shift to larger jobs as the simulation continues. This is clearly not a realistic model of the actual system, so a reasonable arrival process must be devised.

The procedure used here is to start with a queue of 55 jobs selected at random from the pool of known jobs. The total amount of processing needed by these jobs is calculated and stored as a threshold. Whenever the remaining processing in the simulated input queue drops below this threshold, new jobs are selected at random from the unused jobs. When 85% of the total processing of all jobs is complete, the simulation is stopped.

This procedure is preferable to simply returning completed jobs to the input queue because the latter technique would result in overemphasis of the smaller jobs. The 85% stopping criterion avoids the problem of running out of small jobs. The initial queue length, 55, was chosen to maximize agreement between simulated and actual system performance in the validation.

Table 1 presents validation information for the relatively long (morning) measurement period. The actual and simulated system performance measures (first two columns) agree to within 3.5%, with the measures considered most important here (CPU utilization and response time) in agreement to within 1% relative error. The third column displays simulation results with the overhead associated with the software event recorder removed, showing that its effect was a degradation by approximately 2%. The fourth column gives results from a simulation with a different random ordering of the jobs, and again displays relative deviations of about 2%. We conclude from this information that we have constructed a valid and stable simulation model of the actual system.

The short (afternoon) measurement period contains too few interactions and jobs to allow a convincing validation. Table 2 gives the comparison between actual and simulated system performance. The results based on the short period must be taken as showing trends only and as lending credence to the validated data from the longer measurement period.

Table 1  
Validation of Long Measurement Period

	actual system	model system	model system without probe	permuted model
ET <sup>a</sup>	1637.719	1623.747	1578.504	1579.189
CPU <sup>b</sup>	69.73	70.32	72.34	73.30
PPU <sup>c</sup>	82.6	85.5	81.5	83.3
DMP <sup>d</sup>	8.38	8.28	8.33	8.43
Mem <sup>e</sup>	121,880	120,027	115,443	115,346
RT <sup>f</sup>	352	357	336	352

Table 2  
Validation of Short Measurement Period

	actual system	model system	model system without probe
ET <sup>a</sup>	199.172	207.518	202.402
CPU <sup>b</sup>	47.16	45.26	46.40
PPU <sup>c</sup>	96.7	97.0	94.0
DMP <sup>d</sup>	10.17	10.48	10.23
Mem <sup>e</sup>	116,930	114,291	109,791
RT <sup>f</sup>	1373	1092	842

- a) running time in seconds
- b) percentage of CPU utilized by user jobs
- c) average percent of PPU resources utilized
- d) degree of multiprogramming
- e) average memory utilized (out of 131,012)
- f) response time in milliseconds

## IX. Results

Tables 3 and 4 give response times and CPU utilizations, respectively, for each combination of schedulers (S1, S2, and S3) and deadlock control algorithms (D1, D2, D3, and D4). Variability in response times is at least as important as means, because occasional long waits annoy users more than moderate but consistent ones. Therefore both means and standard deviations appear in table 3.

It is immediately apparent that the third scheduler (S3), which selects the cheapest 13 jobs regardless of space constraints and thus overloads memory, performs far worse than the others. None of the deadlock control algorithms was able to cope with the overload. This leads to the conclusion that deadlock control does not serve the purposes of scheduling.

Mean response time, ignoring S3, shows a 563% relative variation (from 330 milliseconds to 2188) while CPU utilization shows only a 22% relative variation (from 64.19% to 77.98%). Thus response time is much more sensitive to the choice of a scheduler or deadlock control algorithm than is CPU utilization.

The preemptive scheduler (S1) yields better response times than the non-preemptive scheduler (S2), but has somewhat worse CPU utilizations. Furthermore, when S2 is used, response times (and particularly their standard deviations) are best for the preemptive deadlock control algorithms (D1 and to some extent D3). It is reasonable to conclude that preemption, especially in the scheduler, is important in interactive systems such as UT-2.

Among the deadlock control algorithms, immediate preemption (D1) gives the best response times and CPU utilizations under both S1 and S2. Detection and recovery (D3) is next best, and complete allocation (D2) performs surprisingly well. However, both are significantly degraded if non-preemptive scheduling is used. Response times under

the avoidance algorithm (D4) are considerably longer than under the others, and the avoidance of algorithm is particularly sensitive to the choice of schedulers.

The CPU utilizations of D3 and D4 reported here are markedly superior to those reported in the preliminary study.<sup>8</sup> This is due both to the use of a model that is more comprehensive in its resolution of job characteristics and to the correction of an invalid implementation of D3 and D4. Precise information on resource requirements, available and utilized in this trace-driven model, is highly favorable to the performance of D2 and D4. Such precise information on resource requirements is not often available in normal production environments.

Statistical techniques<sup>13,14</sup> were used to analyze the contributions of the choice of schedulers (S1 or S2) and deadlock control algorithms (D1 through D4) to the observed variations in mean response time. The computed contributions were: scheduler - 39%, deadlock control algorithm - 35%, and the interaction between them - 26%. The large cross term suggests that the interaction between scheduling and deadlock control is significant in the design of interactive systems. If the third scheduler (S3) is included, the contributions become 70%, 12%, and 18%, respectively. Since S3 is such a poor scheduler, the statistical analysis using it predictably returns distorted results.

Table 3  
Response Time in Milliseconds<sup>a</sup>  
Long Measurement Period

	D1	D2	D3	D4
S1	330(413)	399(690)	361(481)	507(588)
S2	571(1087)	702(2201)	837(1747)	2188(8031)
S3	2106(4361)	7189(15262)	3491(3992)	4933(8505)

a) the numbers in parentheses are the standard deviations of the response time distribution

Table 4  
CPU Utilization (percent)  
Long Measurement Period

	D1	D2	D3	D4
S1	69.32	64.19	67.63	66.81
S2	77.98	71.30	75.85	69.84
S3	32.15	19.47	23.93	21.91

Table 5  
Response Time in Milliseconds  
(Shorter Measurement Period)

	D1	D2	D3	D4
S1	1066(1037)	950(836)	1076(1370)	1196(1209)
S2	1514(2325)	1920(4404)	1618(2691)	2211(4854)
S3	1821(3075)	3427(7192)	4191(4145)	4338(3705)

Table 6

CPU Utilization (percent)  
(Shorter Measurement Period)

	D1	D2	D3	D4
S1	42.31	40.83	40.14	38.83
S2	48.13	45.66	47.03	45.12
S3	41.16	31.25	18.51	15.61

## X. Conclusions

The simulation results support the following conclusions. Deadlock control algorithms cannot cope with the overload produced by an indiscriminate scheduler. Response time is more sensitive than CPU utilization to differences between schedulers and deadlock control algorithms. There is a tradeoff between short response times (produced by preemptive schedulers) and high CPU utilization (produced by non-preemptive schedulers). Preemptive deadlock control algorithms are superior to non-preemptive ones, especially in combination with a non-preemptive scheduler.

## Acknowledgement

This research was supported by the National Science Foundation under grant GJ-1084, "Design and Analysis of Operating Systems".

## References

1. Coffman, E. G., Elphick, M., and Shoshani, A., System deadlocks. Computing Surveys 3, 2 (June 1971), 67.
2. Holt, R. C., On deadlock in computer systems. Ph.D. dissertation, Department of Computer Science, Cornell University, Ithaca, N. Y., January 1971.
3. Shoshani, A., and Coffman, E. G., Detection and prevention of deadlocks. Fourth Annual Princeton Conf. on Information Sciences and Systems, Princeton, N. J., March 1970.
4. Habermann, A. N., Prevention of system deadlocks. Comm. ACM 12, 7 (July 1969), 373.
5. Sherman, S. W., Trace-driven modeling studies of the performance of computer systems. Ph.D. dissertation, Department of Computer Sciences, The University of Texas, Austin, Texas, 1972.
6. Cheng, P., Trace-driven modeling. IBM Systems Journal 8, 4 (1969), 280.
7. Sherman, S. W., Basket, F., and Browne, J. C., Trace-driven modeling and analysis of CPU scheduling in a multi-programming system. Comm. ACM 15, 12 (Dec. 1972), 1063.
8. Sherman, S. W., Howard, J. H., and Browne, J. C., A comparison of deadlock prevention schemes using a trace-driven model. Sixth Princeton Conf. on Information Sciences and Systems, Princeton, N. J., March 1972, p. 604.
9. Thornton, J. E., Design of a Computer: The CDC 6600. Scott, Foresman and Co., Glenview, Ill., 1970.
10. Anon., Control Data 64/65/6600 Computer Systems Reference Manual, Publication No. 60100100, Control Data Corporation, 1967.

11. Howard, J. H., A large-scale dual operating system. 1973 ACM National Conf., Atlanta, Ga., p. 242.
12. Johnson, D. S., A process-oriented model of resource demands in large multiprocessing computer utilities. Ph.D. dissertation, Department of Computer Sciences, The University of Texas, Austin, Texas, 1972.
13. Tsao, R. F., Comeau, L., and Margolin, B. H., A multi-factor paging experiment: I. The experiment and its conclusions. IBM Report RC-3443, July 1971.
14. Tsao, R. F., and Margolin, B. H., A multi-factor paging experiment: II. Statistical methodology. IBM Report RC-3522, September 1971.