1980

Validating System Models: A Case Study

H. D. Schwetman

Report Number:
80-333

Validating System Models: A Case Study

H. D. Schwetman

Computer Sciences Department

Purdue University

West Lafayette, Indiana 47907

CSD-TR 333

Validating System Models: A Case Study

H. D. Schwetman

Computer Sciences Department

Purdue University

West Lafayette, Indiana 47907 USA

Abstract

Validating a model of a computer system means determining that the structure of the model is plausible and that the values of the response variables from the model agree (within some tolerance) with the values of the same variables obtained from the system. In this paper, four models of a single system are described, as are the attempts to validate each of them. Each model is described in terms of its structure, its accuracy and the costs of operation. While no general methodology is developed, facets of these validations can be seen to be applicable in many types of validations.

Introduction

A model is an abstraction of a system which embodies pertinent features of the system. Furthermore, when given inputs representing a system workload, the model produces response variables. Validation of such a model implies that the model meets two criteria:

1. The model is structurally plausible, and
2. In a comparison with the modeled system, the model exhibits similar values of the response variables when presented with an equivalent workload.

This definition, while not rigorous, is intended to be intuitive and to cover many types of system models. Also, this definition may seem obvious (almost trivial), and one should ask, "Why aren't all models validated?". The answer is that, though it sounds attractive and even easy, validation of models of computer systems turns out to be extremely difficult. Upon close examination, there are few published reports of system models which have been validated in this sense.

The goals of this paper are to demonstrate, via four case studies, the difficulties which emerge as validations are attempted and the levels of accuracy which can be achieved. The four models are all abstractions of a large-scale system in use at the Purdue University Computing Center (a CDC 6500). The models are all used to "process" the same workload which was derived from actual data collected during a 2353 second period of operation. With each model, the modeled system, the structure of the model and its workload, the results (values of response variables) and the operating costs are presented. In some cases, errors in the validation as well as the final results are shown. It is hoped that by giving some of these "nitty-gritty" details, other researchers can anticipate some of the problems and the levels of accuracy which can be expected.

The System

The system being modeled is one of two CDC 6500 computers in use a Purdue University. Normally, one of these two interconnected systems emphasizes a time-sharing subsystem and the other emphasizes batch-processing. This study concentrates on the batch-processing system. The pertinent features of this system (mainframe) are:

- a 131072 word main memory (60 bits per word)
- two central processors (CPU)
- ten peripheral or I/O processors (PPU)
- twelve data channels.

The Purdue-MACE operating system can multiprogram up to ten user jobs in main memory. In practice, about seven user jobs and two systems tasks are concurrently active. Each active task requires one control point and a contiguous block of main memory. A preempt-resume job scheduler can select jobs from an ordered queue of waiting jobs for residency in main memory. As the status of these active and/or waiting jobs changes, active jobs can be preempted (rolled out), waiting jobs can be resumed (rolled in), and new jobs can be initiated. All references to main memory are dynamically relocated by the hardware, permitting active jobs to be repositioned in memory as needed, so as to recover free blocks. Also, jobs can be resumed at any location and any control point (or even on the other mainframe) as resources become available.

The CPU's are allocated to active jobs using a round-robin-with-time-slicing strategy. The length of the slice is normally set to 20 milleseconds. Jobs can elect to relinquish use of the CPU while I/O is in progress or to continue to use the CPU in parallel with I/O processing.

All of the file-oriented I/O (disks and tapes) is handled by the peripheral processors. In the Purdue-MACE system, there is a queue of requests associated with each logical device. When an entry (I/O request) is placed in an empty queue, a PPU is loaded with the appropriate driver program and assigned to process the queue of requests. After completing the initial request, any subsequently arriving requests are processed until the queue is finally emptied; at this point, the PPU is returned to the pool of available processors. For historical reasons, these queue processors are called stack processors. Because of this arrangement, delays for I/O experienced by jobs occur at the stack processor level and not at the PPU's; in fact, there is almost no queueing for PPU's in the system described.

The system, described in terms of resources, policies and workload for modeling purposes, consists of:

1. A dynamically allocatable main memory with typically up to seven user jobs active plus two systems tasks.

2. Two homogeneous CPU's,
3. Seven stack processors (associated with seven mass storage devices),
4. A pool of PPU's,
5. The CPU is scheduled using a round-robin with time-slice of 20 msec,
6. The workload is a widely varying collection of jobs, with dynamic arrivals.
7. Jobs are activated in order of highest priority first subject to a constraint of a sufficient amount of available memory; the priority order may be violated if a smaller job down in the queue can be fit in the available memory.
8. Higher priority jobs can preempt lower priority jobs from memory, and
9. File I/O is scheduled using a first-come, first-served (FCFS) policy; file associated with the specified device; file I/O processing can proceed in parallel with CPU and other I/O activities (on separate devices) associated with a job.

Data

The Purdue-MACE system includes a software data-gathering facility [Ewin 74] which can be invoked to collect event data while the system is in operation. These events describe requests, assignments and releases of system resources. For example, every CPU assignment is recorded, showing the time of assignment (in milleseconds), the identity of the task receiving use of the CPU, and the status of the task which had the CPU. These events are recorded at the rate of about 300-500 events per second on the batch-processing systems and nearly 1000 events per second if the time-sharing subsystem is active.

For the purposes of the current study, a 2353 second period of normal production was observed, starting at 14:31 hours on Friday, March 3, 1978. During this time, 190 different tasks were seen, 614 tasks were rolled out, some more than once, and 169 jobs were completed (or were active when the observation period stopped).

While an enormous amount of information is available in this file of events, a subset of items was extracted and is shown in Table 1. The selection of items was guided in part by knowing the data requirements for the models which were being developed.

i	Resource	Assignment Count C_i	Busy Time B_i	Waiting Time W_i
1	CPU (2)	245210	4051.267 sec	6897.774 sec
2	SP - 0	1504	71.506	77.797
3	SP - 2	2087	194.056	239.083
4	SP - 3	3552	283.470	313.130
5	SP - 4	14030	979.516	1623.902
6	SP - 9	12472	1012.528	1759.818
7	SP - T	11599	568.548	831.637
8	PPU	18022	1706.324	1706.324
	Control points	783	21225.807	

Notes: Elapsed time (ET) = 2353.379 sec

SP - i means stack processor for channel i, T means tape channel

Waiting time is queueing time plus service time

Table I

Summary of System Data

Using the notation of operational analysis [DenB77], it can be seen that the following response variables for each resource can be calculated from the data in Table I as follows: for resource i

mean service time - $s_i = B_i/C_i$,

utilization - $U_i = B_i/ET$,

mean queue length - $\overline{n}_i = W_i/ET$,

mean waiting time - $R_i = W_i/C_i$,

Throughput rate - $X_i = C_i/ET$.

The values of these responses for the eight major resources of the system are shown in Table II.

i	Resource	S_i	U_i	$\overline{n}_i$	R_i	X_i
1	CPU (2)	.017 sec	1.721(.86*)	2.931	.028 sec	104.194 tasks/sec
2	SP - 0	.048	.030	.033	.052	.639
3	SP - 2	.093	.084	.102	.115	.887
4	SP - 3	.080	.120	.133	.088	1.509
5	SP - 4	.070	.416	.690	.116	5.962
6	SP - 9	.081	.430	.748	.141	5.300
7	SP - T	.049	.242	.353	.072	4.929
8	PPU (7)	.095	.104	.725	.095	7.658

* server utilization

Table II

Response Variables from System Data

The Network-of Queues Model

The first model is a variation of Buzen's Central Server Model [Buz 73], shown schematically in Figure 1. Similar models have been widely used and discussed and, consequently, will not be described here, except to point out the presence of two CPU's and the two CPU feedback paths. The slice-end path is required to account for those CPU intervals which terminate because the end of a time slice is reached (and not because an I/O process is to begin or a program is finished).

The parameters required for this model are the mean service times, s_i, the branching probabilities, p_i, and the level of multiprogramming, mp. The service times are shown in Table II. The branching probabilities can be approximated using the branching frequencies calculated from the assignment counts. The number of transitions along the time-slice-end path is assumed to be all exits from the CPU which could not be tagged as either an I/O process, a system PPU request, or a job termination. Using 245,210 CPU assignments, the number of time-slice ends is estimated to be 181,161. The parameter values for the model are presented in Table III. In Table II, the disciplines refer to the service disciplines used at each device. These are restricted to the four disciplines listed in [BCMP 75]. Notice that a program here is really a segment of code which terminates either for a preemption or a completion.

These parameters were used as input to a network-of-queues solution program available at Purdue [BBS 77]. The selection of the level of multiprogramming is not a clear choice. If the total assignment time for the control points is divided by the elapsed time, we can conclude that the average level of multiprogramming is 9.019 tasks. However, three of these tasks are systems tasks (the system, the SPOOL'ing package, and the event recording task) and really do not contribute to the load very much, giving an effective level of multiprogramming of 6.019 tasks. Another way to estimate this parameter is to sum the $\overline{n}_i$ values from Table II, giving an estimated level of multiprogramming of 5.715 tasks. The validity of this estimate is based on the assumption that there is little overlapped usage of resources within individual tasks. In practice, this is a reasonable assumption with levels of multiprogramming greater than about two tasks.

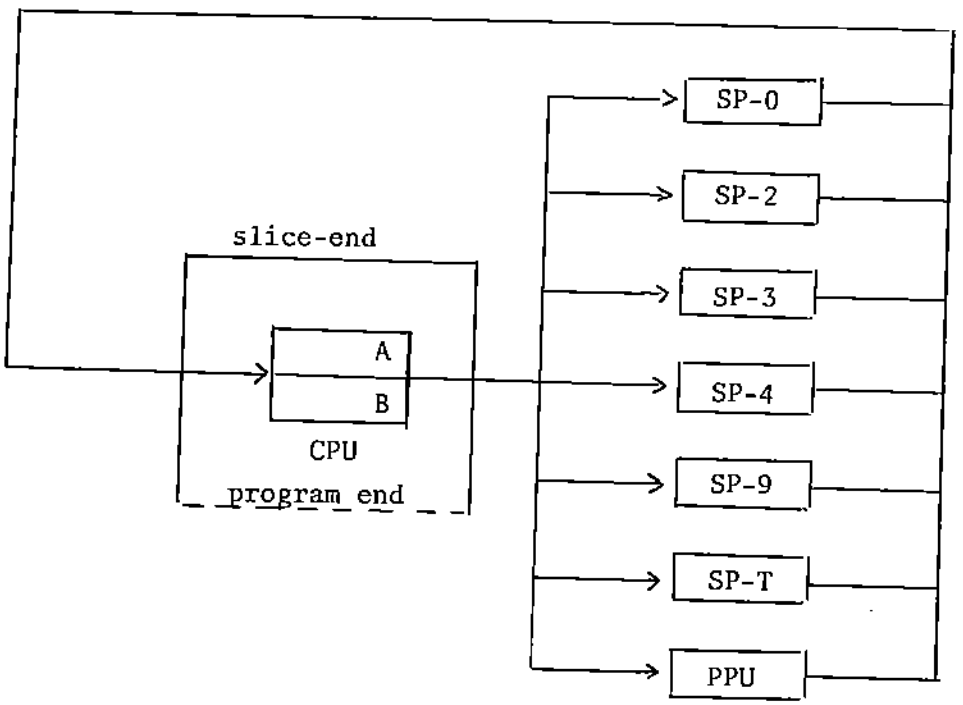


Figure 1

Schematic of Network-of-Queues Model

The output of the network solver is summarized in Table IV for levels of multiprogramming of five, six and seven tasks, along with comparable values from the system data. When comparing these results, a note of caution is in order. With multiserver devices, such as the CPU, device utilizations produced by the network solver are given by 1 - pr (device idle), while in the system data, utilizations are given by dividing the device busy time by the elapsed time. The network solver does produce a server utilization (given in Table II) which is calculated using Little's Law to obtain the expected number of tasks in service and then dividing by the number of servers.

It is difficult to assign a figure of merit to the results in Table IV. It can be seen that of the 32 data items calculated, 19 items fall between the solution values for levels of multiprogramming of five and six. It is instructive to notice that all of the utilizations and device throughputs were "correct", whereas many of the waiting times (5 out of 8) and queue lengths (6 out of 8) were in error. However, these errors were often not of great magnitude. Evidently, queue lengths and waiting times are more sensitive to the assumptions of the model (e.g. exponential service time distributions) than are utilizations and throughput rates are related so that accurate utilizations guarantee accurate throughput rates and vice versa.

The costs of solving this model for 10 levels of multiprogramming were minimal, given that the input parameters had been prepared. The estimated costs of obtaining these 10 solutions are as follows:

CPU seconds	2.979	$.05
1.0 units	300	.02
Memory charge		.06
Print lines	619	.11
Total		$.24

using the charging scheme used on the CDC 6500 systems at Purdue.

i	Resource	Discipline	S_i	P_i	Counts
1	CPU (2)	Processor-sharing	.017	.73880+ .00319	181161 783
2	SP - 0	FCFS	.048	.00613	1504
3	SP - 2	FCFS	.093	.00851	2087
4	SP - 3	FCFS	.080	.01449	3552
5	SP - 4	FCFS	.070	.05722	14030
6	SP - 9	FCFS	.081	.05086	12472
7	SP - T	FCFS	.049	.04730	11599
8	PPU (7)	Infinite servers	.095	.07350	18022

Table III

Parameter Values for Network-of-Queues Model

	Utilization				Queue Length				Waiting Time				Throughput Rates			
	5	Sys	6	7	5	Sys	6	7	5	Sys	6	7	5	Sys	6	7
CPU	.85	.86	.91	.94	2.57	2.93	3.29	4.08	.026	.028	.031	.037	99	104	106	111
SP-0	.03	.03	.03	.03	.030	.033	.032	.034	.049	.052	.049	.050	.61	.64	.65	.68
SP-2	.08	.08	.08	.09	.085	.102	.091	.096	.100	.115	.101	.102	.85	.89	.91	.95
SP-3	.12	.12	.12	.13	.112	.133	.139	.146	.089	.088	.090	.091	1.4	1.5	1.5	1.6
SP-4	.40	.41	.43	.45	.591	.690	.679	.746	.104	.116	.111	.118	5.7	5.9	6.1	6.4
SP-9	.41	.43	.44	.46	.616	.748	.709	.781	.122	.141	.131	.138	5.1	5.3	5.4	5.6
SP-T	.23	.24	.25	.26	.287	.353	.318	.339	.061	.072	.063	.065	4.7	4.9	5.0	5.3
PPU	.10	.10	.10	.11	.694	.725	.744	.775	.095	.095	.095	.095	7.3	7.7	7.8	8.2

Table IV

Solutions from Network Solver

One key descision to be made in constructing a network-of-queues model is the choice of resources to be included. Other possible formulations of models of the Purdue system could be based on using the CPU's and the data channel's or the CPU's and the pool of PPU's as the two types of resources required for the central server model. An analysis of the data reveals, however, that the sums of the queue lengths (4.812 for the channel model and 5.164 for the PPU model) do not account for the active tasks as well as the stack processor model given above. Furthermore, the structures of these models do not agree with our conceptual view of the system.

Asymptotic Performance Bounds Using Operational Analysis

One new approach to analyzing system data and predicting performance is to estimate system performance for one job active in the system and then to estimate performance when the system is saturated. This approach is described in [DenB77]. For a central server model, this analysis can be done using the data presented in Table I. The approach depends on determining the total time spent at each device by a single "average" task. In saturation, one or more devices is fully utilized and, in effect, controls the system throughput rate. It can be shown that the saturating device is the one at which tasks spend the maximum time, conversely the device with the slowest throughput rate in saturation.

We can convert the earlier network-of-queues model to a central server model by eliminating the time-slice-end path and then correctly accounting for the two servers in the CPU facility. It is clear that the expected number of visits to the CPU per I/O request or program termination, n_c, can be expressed as follows:

let a = the probability of a time-slice-end, then

$$n_c = \sum_{i=1}^{\infty} a^{i-1}(1 - a) = 1/(1-a).$$

Thus, for the system data, $n_c = 1/(1 - .73880) = 3.82848$ is the expected number of visits to a CPU per entry to the CPU facility. This gives a corrected CPU service time of $.017n_c = .065$ seconds. The corrected CPU visit count becomes 64049.

The number of visits to device i per task entry to the system, denoted V_i, is given as: (note that i = 0 denotes the "exit" device)

p_0 = probability of exit,

$V_1 = 1/p_0$ = CPU visits per task, and

$V_i = p_i V_1$, for $i > 1$.

The total time at device i then becomes $V_i s_i$. The calculations required to produce these for each device are given in Figure 2.

	0	1	2	3	4	5	6	7	8
C_i	783	64049	1504	2087	3552	14030	12472	11599	18022
P_i	.01233	0	.02348	.03258	.05546	.21905	.19473	.18110	.28127
V_i	1.0	81.1	1.9	2.6	4.5	17.8	15.8	14.7	22.8
s_i		.065	.048	.093	.080	.070	.081	.049	.095
$V_i s_i$		5.272	.091	.246	.360	1.244	1.279	.720	2.167

Figure 2

Calculation of Asymptotic Bounds

The throughput rate for the system with one active task, $X_0(1)$, is the sum of the total device times:

$$X_0(1) = [R_0(1)]^{-1} = \left[\sum_{i=1}^{nd} V_i s_i\right]^{-1} = .088 \text{ tasks per second.}$$

In saturation, the CPU with two busy servers has an estimated throughput rate of $[5.272/2]^{-1} = .376$ tasks per second. The PPU's, which could have upto seven servers busy will have an estimated throughput rate of $[2.167/7]^{-1} = 2.1$ throughput rate is bounded from above by .376 tasks per second. The throughput rate observed in the data is 783/2353.379 = .333 tasks per second, indicating that the system is operating at near saturation. These bounds, as well as the throughput rates for one to ten tasks active as derived from the solution of the network, are summarized in Figure 3.

The costs associated with this type of analysis are nil, assuming that the data values are available, since only a calculator was used.

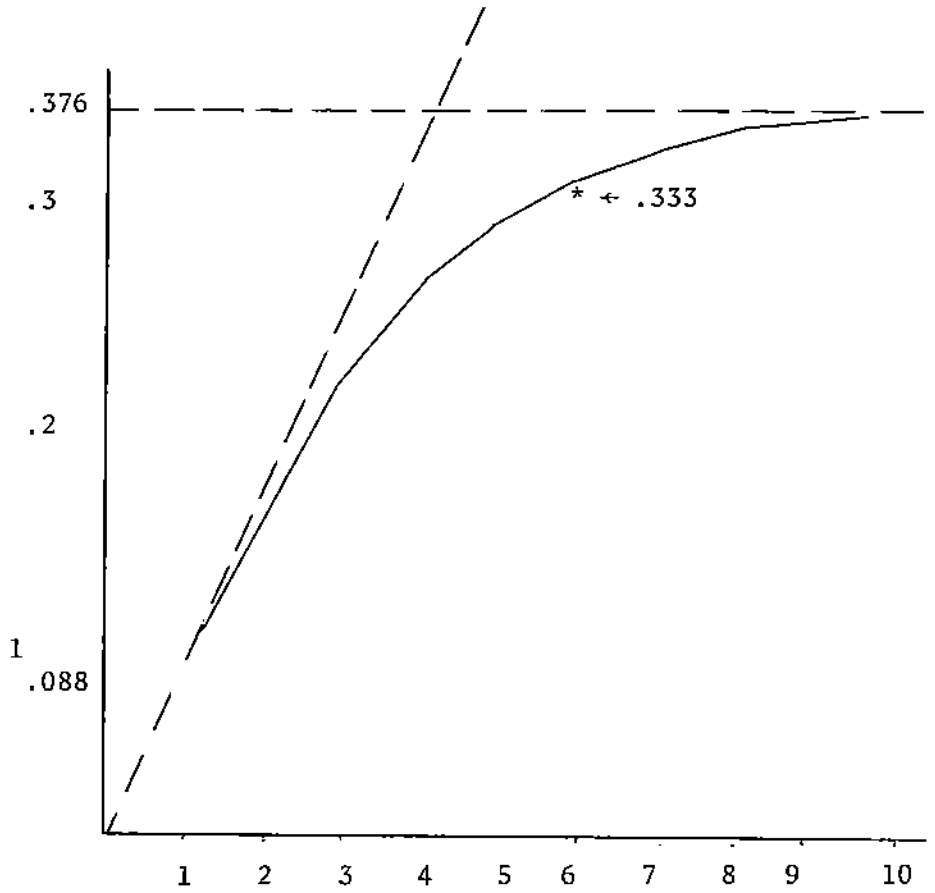


Figure 3

System Throughput Rates versus Level

of Multiprogramming

Trace-driven Simulation Models

Discrete event simulation is another modeling technique which can be used to predict the performance of a computer system. The basic structure of such a model has been described in [Mac 70]. In addition to estimating the response variables mentioned above, this type of model can be used to study job related statistics such as job turnaround times and job in-memory or residency times; here, a job consists of one or more segments (periods of activation). Furthermore, since it is possible to include additional resources in the model, e.g. main memory and control points, it is feasible to use this type of model to study job selection strategies as well as the effects of changes in these resources. A further use of such a model is to evaluate the effects of changes in the model which are not realizable in the earlier models, e.g. using an I/O scheduling policy other than FCFS.

One problem with simulation modeling is determining an adequate description of the workload. One approach depends on deriving probability distributions to describe the patterns of demands for service made by the jobs in the workload. Another approach describes the workload as a sequence of demands contained in a file. This file is derived from system event data and is called a trace file. This approach has been described in several articles, including [SBB 72]. An elaborate trace driven simulation model (TDM) of the Purdue system has recently been developed [MeaS 78].

In this model, a file of event data is processed to create a set of job scripts, where a job script is the list of all demands for resources made by a single job. The companion simulation model activates jobs, using a selection strategy similar to the one used in the system and then reads the file of job scripts for each of the active jobs to control the order of resource requests and the service times for each request. The model uses statistically derived distributions to model the load imposed by systems tasks. CPU-I/O overlap is approximated by permitting overlap on write operations and prohibiting overlap on read operations. The Ph.D thesis by Mead [Mead 78] describes this model and several variations on the concept of using job scripts to drive simulation models.

Tables V and VI present data comparing the performance of the system and the performance from the trace driven model. Examination of the data shows that many of the response variables from the two systems exhibit fairly close agreement. It is questionable as to whether the trace driven model can be considered to be a valid model of the system. The basic problem appears to be that there are not enough demands for resources in the TDM; in fact there appears to be at least one fewer jobs in the active state in the TDM.

One goal for developing the TDM was to test various workload characterization techniques. One response variable of interest is the mean job residency time - the amount of time a job is active (in main memory). This is an important variable because it is sensitive to both the resource demands by jobs as well as the interference in accessing resources caused by other active jobs. A comparison of the residency times observed for jobs in the real system and for the simulated system was made. This comparison was based on the correlation coefficient for the two sets of residence times and a scatter plot for all 759 segments observed in the data (here, a segment is that portion of the jobs activity which begins and ends with a change in the amount of main memory required). The data for this comparison performed by Mead is summarized in Table VII and Figure 4. These data show that the TDM models job demands for resources and job interactions very accurately.

The costs of operating the trace driven model depend on the level of detail incorporated into the model and in the workload description. Table X (in the next section) summarizes the costs of operating the TDM described here.

	C_i		B_i		W_i	
	System	TDM	System	TDM	System	TDM
CPU	245210	155458	4051.267	3976.617	6897.774	5735.277
SP-0	1504	1576	71.506	113.453	77.797	118.742
SP-1		60		33.500		33.500
SP-2	2087	1973	194.056	186.709	239.083	227.183
SP-3	3352	3225	283.470	289.318	313.130	333.168
SP-4	14030	13748	979.516	824.976	1623.902	1142.335
SP-9	12472	11863	1012.528	727.863	1759.818	1070.347
SP-T	11559	8738	568.548	563.024	831.637	693.089
PPU	18022	12877	1706.324	1236.190	1706.324	1236.324
Control Points	783	21225.807				

Table V

System Data and Trace Driven Model Data

	S_i		U_i		$\bar{n}_i$		R_i		X_i	
Response	Sys	TDM	Sys	TDM	Sys	TDM	Sys	TDM	Sys	TDM
CPU	.017	.026	.860	.845	2.931	2.437	.028	.037	104.195	66.068
SP-0	.048	.072	.030	.048	.033	.050	.052	.075	.639	.670
SP-1		.558		.014		.014		.558		.025
SP-2	.093	.095	.084	.079	.102	.097	.115	.115	.887	.839
SP-3	.080	.090	.120	.123	.133	.142	.088	.103	1.509	1.371
SP-4	.070	.060	.416	.351	.690	.485	.116	.083	5.962	5.843
SP-9	.081	.061	.430	.309	.748	.455	.141	.090	5.300	5.042
SP-T	.049	.064	.242	.239	.353	.295	.072	.079	4.929	3.714
PPU	.095	.096	.104	.075	.725	.525	.095	.096	7.658	5.473
					5.715	4.500				

Table VI

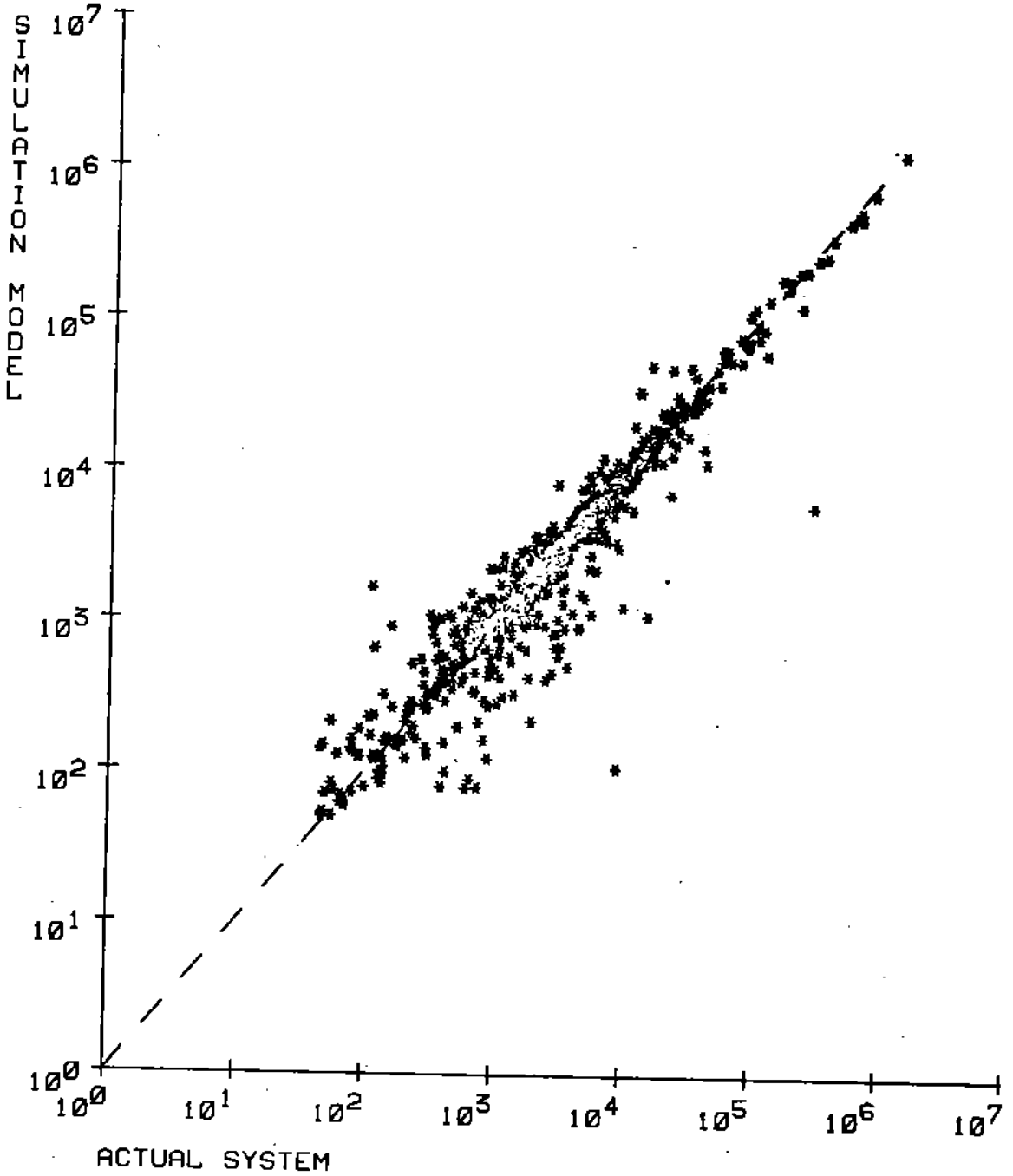
Response Variables from System Data and Trace Driven Model Data

	System	TDM
Mean Job Residency Time	58.751 sec	51.673 sec
Mean Segment Residency Time	14.353	12.663
Correlation Coefficient	.986	

Table VII

Comparison of Residency Times

(System and Trace Driven Model)



Log-Log Plot of Segment Residency Times

(time expressed in milliseconds)

Figure 4

SRT Plot for Production Workload

Hybrid Simulation Models

A modeling technique which combines descrete-event simulation and analytic modeling has been developed [Sch 78]. In such a hybrid model, the resources are partitioned in two groups: long term resources such as main memory and control points, and short term resources, such as CPU's and I/O processors. Discrete event simulation is used to model requests and assignments of long term resources, producing a mix (collection) of active jobs. Usage of the short term resources is then approximated by using an estimated execution time for each active job. These execution times are obtained from some form of an analytic model which uses information about all of the active jobs to estimate the residency time for each job. The goal of this type of model is to achieve accuracies which are comparable to an equivalent simulation-only model, but with greatly reduced operating costs.

In the hybrid model of the Purdue system, a job's demands for short term resource are expressed in terms of cycles or trips around a central server model. The job-script programs described in the preceding section were used to classify each job segment into one of three job classes, based on the mean CPU service interval for the segment. The three classes thus consist of jobs with either short CPU intervals (0 - 20 milleseconds), medium intervals (21 - 100 milleseconds), and long intervals (greater than 100 milleseconds); the mean intervals for each of these classes were 5, 38 and 457 milleseconds respectively. By processing the jobs scripts and classifying the segments into these three classes, we can produce a list of job segments, which contains for each segment a class designator, the number of cycles requried, and the main memory required. At the same time, the input parameters for a three-class network-of-queues models are created. The network solver program is then used to solve this model for all combinations of jobs from three classes which result in levels of multiprogramming ranging from zero to eight jobs. The solutions are in the form of the expected cycle time for jobs of each class for each combination of jobs. The hybrid model then uses the list of segments and the expected cycle times to simulate operation of the system.

Earlier work [Sch 78] established that the hybrid technique could achieve the same levels of accuracy as found in the equivalent simulation-only model, at significant reductions in operating costs. However, the validity of this technique in modeling actual system has not been established. Table VIII

Resource	U_i Sys	U_i Hyb	$\bar{n}_i$ Sys	$\bar{n}_i$ Hyb	R_i Sys	R_i Hyb	X_i Sys	X_i Hyb	S_i Sys	S_i Hyb **
CPY*	.860	.808	2.931	2.303	.107	.103	27.251	22.301	.065	.005,.038,.457 (.072)
SP-0	.030	.029	.033	.033	.052	.047	.639	.632	.048	.046
SP-2	.084	.050	.102	.052	.115	.105	.887	.500	.093	.099
SP-3	.120	.060	.133	.074	.088	.090	1.509	.823	.080	.084
SP-4	.416	.242	.690	.315	.116	.071	5.962	4.409	.070	.055
SP-9	.430	.156	.748	.185	.141	.050	5.300	3.715	.081	.042
SP-T	.242	.240	.353	.297	.072	.080	4.929	3.689	.049	.065
PPU	.104	.128	.725	.897	.095	.105	7.658	8.542	.095	.105
			5.715	4.153						

Table VIII

Comparison of System Data and Hybrid Data

* Here the mean CPU service interval is .065 (see Figure 2) for the system the mean CPU service interval for Hybrid model is .072.

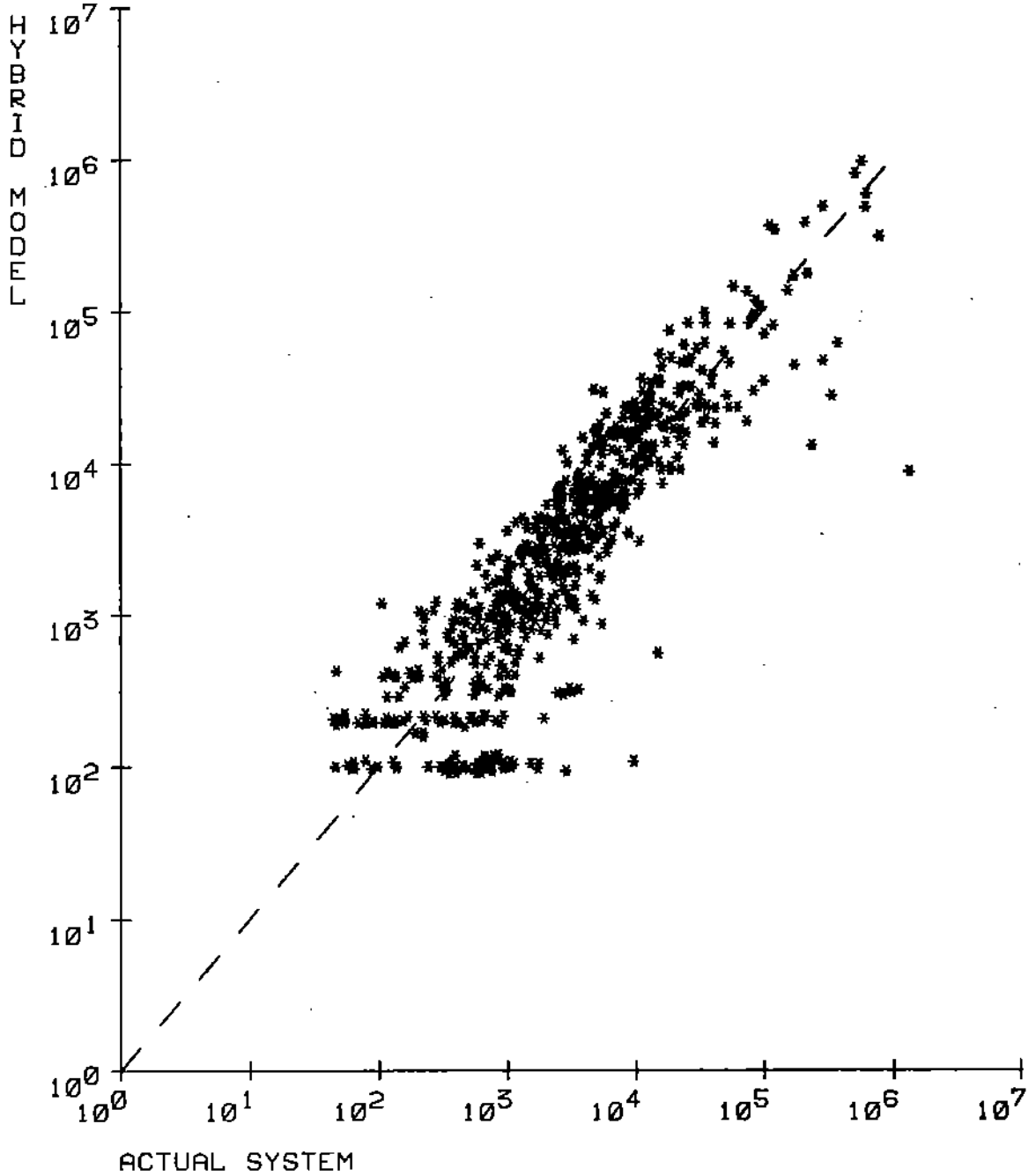
** Derived from scripts (without system tasks)

contains data which can be used to compare the values of the response variables from the hybrid model with those from the system. It can be seen that the hybrid model does not accurately model the system and thus cannot be considered as a valid model.

An examination of the structure of the hybrid model shows that some important features of the system were omitted. The most critical omission is the pair of systems tasks which were present in the system. Mead [Mead 78] showed, using trace driven modeling, that these systems tasks were necessary if an accurate model of the system was to be achieved. Another issue is the ability of the hybrid model to accurately model the behavior of individual jobs. The correlation coefficient of the segment residency times from the system data and the hybrid model is .633, and the scatter plot of these residency times is presented as Figure 5. Further analysis by Mead showed that five or more classes (where classes are formed as described here) are required to produce segment residency times which are of satisfactory accuracy. He also presented a three-class simulation model and reported levels of accuracy similar to those of the three-class hybrid model. These findings are summarized in Table IX. These data suggest that the inaccuracies in the hybrid model may be due to limitations in the input data and not to the hybrid technique.

A natural extension to the hybrid model of the Purdue would be to incorporate more job classes into the model. Unfortunately, as the model is currently implemented, the model's storage requirements to become unacceptably large, when more than four classes are used. Also, the execution costs of the network solver increase drastically as the number of classes is increased. A better technique for calculating expected cycle times seems to be required before the hybrid model can become valid.

The costs of using the hybrid model are compared with the trace driven model and the three-class simulation model in Table X. It can be seen that the hybrid model is much less expensive to use.



Log-Log Plot of Segment Residency Times

(time expressed in milliseconds)

Figure 5

	Elapsed Time	Job Resid. Time	Correl. Coeff.
System	2353 sec	58.571 sec	
TDM - scripts	2353	51.673	.986
TDM - 3 classes	2414	51.719	.613
Hybrid - 3 classes	2366	52.901	.633

Table IX

Comparison of Job Reisdency Times

	CPU Time	I/O Units	Total Cost
TDM - scripts	831 sec	17284	$29.99
TDM - 3 classes	653	1573	22.75
Hybrid - 3 classes			
Solving Network	64	736	2.24
Producing Tables	18	899	.61
Executing Model	28	647	1.28

Note: The cost of preparing the scripts and the segment lists is not included.

Table X

Costs of Simulation Models

Discussion

Normally, the purpose of forming a model of a computer system is to use the model to predict system performance as changes are made in the operating environment or the system's resources. For these predictions to be believable, the model must be validated. In this paper, two criteria have been used to judge the validity of four different models of the CDC 6500 system in use at Purdue University. The criteria require that the structure of the model conform to a conceptually valid view of the system (plausibility) and that the values of the response variables from the model "match" the values from the modeled system. There are situations in which there are no "real" response variable values to be used in a validation (e.g. a model of a nonexistent system); in this case, the plausibility criterion is the only criterion which can be applied in a validation of the model.

The four models presented in this paper were all judged in terms of plausibility and accuracy. In all of these models, simplifying assumptions were made in the implementation of the model. In spite of these, the network-of-queues model and the asymptotic-behavior model gave accurate results. The trace driven model and the hybrid were more ambitious in terms of information produced, but gave less accurate results. The hybrid model was probably too simple to accurately model the Purdue system, but the technique is of interest, because of its low operating costs.

The sensitivity of these models to the simplifying assumptions is unpredictable. One crucial issue is the choice of resources to be included in the model. A bad choice can produce a model which cannot be validated. The other crucial area in modeling building is the gathering of input parameters and test values for the response variables. Projects which have access to "live" sources of data should be able to achieve more valid models.

This paper has not provided a general validation methodology. Denning and Buzen [Den 77] do present such a methodology in very general terms; they do not discuss specific points which would apply to specific models. In the current project, the first accuracy check is to compare the utilizaitons. If these do not agree, then further checking is probably not warranted; the model is probably not valid. The second checks are of the mean queue lengths and the means waiting times at the major resources. The sum of the mean queue lengths could give insight into the level of multiprogramming which was achieved (if overlapped usage of resources is not a major feature

of the model). After these checks are made, then other comparisons can be made, depending on the features which are implemented in the model and the data which is available from the system. The correlation coefficients of the segment residency times in the trace driven model are an example of this type of analysis. As the famous philosopher once said; "Accurate CPU utilizations do not a valid model make".

Acknowledgements: The author must acknowledge the vital contributions made by Steven Bruell and Gianfranco Balbo, for their work on the network-of-queues solver, and to Bob Mead, for his work on the job script generator and the trace driven model.

List of References

BBS77 Balbo, G., Bruell, S., and H. Schwetman, Customer classes and closed network models - a solution technique, Proc. IFIPS 77, North-Holland Pub., 1977, p. 559-564.

BCMP75 Baskett, F., Chandy, C., Muntz, R. and F. Palacios, Open, closed and mixed networks of queues with different classes of customers, JACM (22,2), Apr 1975, p. 248-260.

Buz73 Buzen, J., Computational algorithms for closed queueing networks with exponential servers, CACM (16,9), Sep 1973, p. 527-531.

DenB77 Denning, P. and J. Buzen, Operational analysis of queueing networks, Proc. 3rd International Symposium on Modeling of Computer Systems, 1977, p. 151-172.

Ewi74 Ewing, J., CPUMTR Software Event Probe, Purdue University Computing Center Document LO-EVNTPE, Dec 1974.

Mac70 MacDougall, M. Computer system simulation: an introduction, Computing Surveys (2,3) Sep 1970, p. 6780684.

MeaS78 Mead, R. and H. Schwetman, Job scripts: a workload description based on event data, Proc. NCC 78 (AFIPS), p.

Mead78 Mead, R. On the Modeling of Resource Demands in a Multiprogrammed Computer System, Ph.D. Thesis, Purdue University, 1978.

SBB72 Sherman, S., Baskett, F. and J. Browne, Trace driven modeling and analysis of CPU scheduling in a multiprogramming system, CACM (15,2), Dec 1972, p. 1063-1069.

Sch78 Schwetman, H., Hybrid simulation models of computer systems, CACM (to appear Sep 1978).