W. Chiu
D. Dumont
R. Wood

Performance Analysis of a Multiprogrammed Computer System

Abstract: A combination of analytical modeling and measurement is employed for the performance analysis of a multiprogrammed computer system. First, a cyclic queue model is developed for the system under study. Then, model validation is attempted in both controlled and normal environments. The success of the model is demonstrated by its prediction of performance improvements from system reconfigurations. Reasonable correlation between the measured performance and the model predictions under various degrees of multiprogramming is observed. Finally, possible system reconfigurations are explored with the insight gained from the performance analysis.

Introduction

Computer performance evaluation is a topic of considerable current interest. In general it deals with the problems of (1) system comparison, (2) system tuning, and (3) system design; see, for example, [1, 2]. Two evaluation techniques are commonly employed: the measurement of actual systems and modeling, which includes simulation, statistical techniques, and analytical methods. Measurement yields the greatest fidelity with respect to actual configurations; the analytic approach is most attractive because it can economically provide general insights into computer operations. Much effort has been directed towards both analytical modeling and performance measurements [1, 2, 3], but too often these efforts have been decoupled.

The analytical models which have been constructed are usually gross approximations of actual systems, often containing simplifying assumptions for the sake of mathematical tractability. The complaint about these models is that they are too simple to help the performance analysis of real-life systems; therefore, the success of these models depends largely on how closely they can predict the performance of actual systems. Efforts are currently being made to analyze more realistic models. However, measurements are often made without a structural model as guidance; thus, they cannot be easily analyzed and any conclusions drawn are difficult to generalize to other systems or even to small system changes. In this paper we attempt to combine the two techniques, analytical modeling and measurements, for the performance analysis of a multiprogrammed computer system. First, a cyclic queue model (CQM) is developed for the system under study and, second, we discuss the validation of the CQM in both controlled and normal environments. Then we compare the results from the CQM with the results obtained from another model, the central server model (CSM). Finally, possible system reconfigurations are explored with the insight gained from the performance analysis.

Model development

The development of a model for an actual computer system depends on the structural and operational aspects of the system and on the data desired. The hardware modules in the system define its structure. The workload characteristics, the operating system strategies, and the performance of the hardware modules define its behavior. Each of these aspects is examined in detail in the following paragraphs.

The computer system being modeled is the IBM System/360–75 at the University of California, Santa Barbara (UCSB). The major components of this system are the CPU, the main memory, consisting of 512 kilobytes of high-speed core, and two megabytes of bulk core, as well as two high-speed selector channels, and a multiplexor channel. Auxiliary storage consists of two IBM 2314's (eight disk drives on each selector channel) and an array of tape drives and hard copy equipment (on the multiplexor channel).

The operating system is the standard 360 OS/MVT with HASP [4] that performs the spooling function, modified to service a time-sharing system (UCSB on-line system) developed at UCSB. The principal focus of this study is

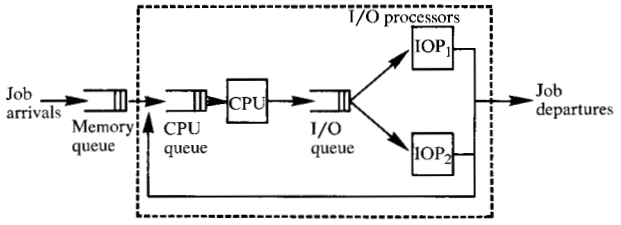

Figure 1 Job flow.

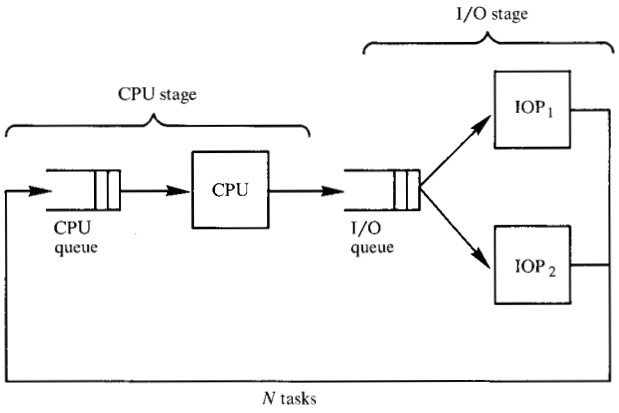

Figure 2 Cyclic queue model.

on the workload generated by the batch jobs, although the impact of the on-line system on this workload is also considered.

A job can be broken up on several distinct tasks (known as steps), such as COMPILE, LINK-EDIT, and EXECUTE, which must be executed sequentially. Along with these steps, system tasks are also invoked, for example, reader/interpreter, initiator/terminator, etc. These tasks are considered to be part of a job's execution [5].

Job I/Os to hard copy devices are spooled on auxiliary storage (disks). The actual I/Os to these devices are performed by HASP. They are completely overlapped with CPU processing (buffered) and require very little CPU time; therefore, these devices and their channel (multiplexor channel) can be excluded from the model. Tape operations also utilize the same channel, but the small amount of activity observed on our system warrants their exclusion from the model.

A disk I/O involves both the channel and a disk drive. Drives in the same disk unit can be busy simultaneously, but this occurs rarely in our system. Thus, there are two I/O processors in the model, each I/O processor (IOP) consisting of both a selector channel and a disk unit.

The model shown in Fig. 1 describes the job flow through the system. Jobs arrive and wait in the input (or memory) queue. When a job enters main memory and becomes active, it competes for the CPU and the IOPs. The measure of throughput is the rate of CPU service request completions, which is proportional to the job completion rate. Under heavy traffic conditions we may represent the system with a closed model, as shown in Fig. 2. Such a model is called a cyclic queue model; its analysis is discussed later. It assumes a constant number of tasks, N, circulating in the system. A job in execution may be characterized by an alternating series of CPU requests and I/O service requests until completion. The degree of multiprogramming may vary with dynamic partitioning (OS/MVT); however, as an approximation an average number, N, is used in the model. The model has one I/O queue for the two I/O processors instead of individual I/O queues. This simplifying assumption is tolerable because the resource allocator tends to balance the load on the two I/O processors. Reference to a model with separate I/O queues will be made during model validation. In the model a task is not allowed to overlap its compute and I/O operations, even though the actual system does have facilities to coordinate overlapping compute and I/O operations of the same task. Measurements on a System/360 Model 91 reported in [6] indicate that such overlapping was seldom employed, in fact less than five percent for their workload; we believe our system should behave similarly.

When the CPU is busy, CPU requests must wait in the CPU queue, where the order of task selection is based on priority. The compute time for each task request may be characterized by a random variable, independently and identically distributed (i.i.d.) for all tasks. The I/O service time is also characterized by an i.i.d. random variable for all tasks.

The model shown in Fig. 2 is similar to the machine-repair problem studied by Koenigsberg [7]. This problem is analyzed as a system of cyclic queues, thus the name cyclic queue model (CQM).

For the computer model, tasks are analogous to machines, with the I/O processors representing the working stations and the CPU the only repairman. When the number of I/O processors equals or exceeds the number of circulating tasks (i.e., no I/O queuing occurs), and when the I/O time and compute time distributions are both exponential and the task scheduling at the CPU and I/O queues is first-come-first-served (FCFS), the model is known as the machine interference problem [8, 9]. The I/O service time probability density function is given as

$$f_{\mathrm{I/O}}(t) = \lambda e^{-\lambda t} \tag{1}$$

with mean $1/\lambda$, and the CPU processing time probability density function is given as

$$f_{\mathrm{CPU}}(t) = \mu e^{-\mu t} \tag{2}$$

with mean $1/\mu$. This is also a special case of the queuing network model analyzed by Jackson [10] and by Gordon and Newell [11].

The steady state CPU utilization is given as

$$p = 1 - \frac{1}{\sum_{n=0}^{N} \rho^n \frac{N!}{(N-n)!}}, \quad (3)$$

where $\rho = \lambda/\mu$, the load factor, and N is the average degree of multiprogramming.

When the number of I/O processors is less than N, the steady state CPU utilization is given as

$$p = 1 - \frac{1}{\sum_{n=0}^{N-I+1} (I\rho)^n + (I\rho)^{N-I+1} \sum_{k=2}^{I} \left[\rho^{k-1} \prod_{j=1}^{k-1} (I-j) \right]}, \quad (4)$$

with I equal to the number of I/O processors. The average system throughput rate is therefore given by

$$R_c = p\mu, \quad (5)$$

which is the mean number of task compute time (CPU service request) completions per unit time.

Another specialization of the queuing network model is the central server model (CSM) [12]. The model represents the case where there is a separate I/O queue for each IOP. Each task completing a compute time has a certain probability of entering an I/O queue. Both the CQM and the CSM results are compared.

Measurements from our system have shown that compute time distributions are not exponential and that the coefficient of variation (CV) is often greater than 1 (for exponential distributions CV = 1); the empirical data are more closely approximated by hyperexponential distributions, i.e., a weighted sum of exponentials [12, 13]. An example is shown in Fig. 3; see [15] for an analysis of a cyclic queue model with hyperexponential compute times. In general its effect is degradation of resource utilization.

CPU scheduling in our system is preemptive-priority-driven, known as the heuristic approach [4, 16, 17, 18], and not first-come-first-served. Higher priority is generally given to I/O-bound tasks or tasks with extremely short compute times. Priorities are updated periodically based on some function of past behaviour. When a higher priority task completes an I/O request it preempts a lower priority task that is using the CPU. The lower priority task will resume execution only when no other task of a higher priority is requesting the CPU. The result of this scheduling policy is such that for CPU utilization computations we can approximate the CPU stage with a first-come-first-served CPU scheduling and an exponential compute time distribution of the same mean. See [15] for a detailed verification of this assertion. Our empirical data also show that this is a fairly good approximation. Figure 4 is a diagram of the data for the same workload as that of Fig. 3 except that heuristic CPU scheduling is employed in Fig. 4. The data shown are the cumulative distribution of compute time between consecutive I/O requests (each continuous compute time is not necessarily contributed by the same task). The smooth curve is the exponential.

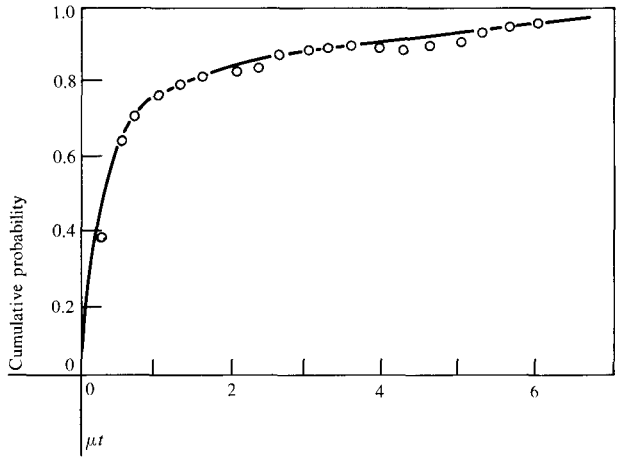

Figure 3 Empirical data of compute time approximated by a hyperexponential distribution.

The distribution for I/O times is also assumed to be exponential. Figure 5 shows the data fitted with an exponential distribution. As can be seen, the fit is not good but at least the first two moments can be matched with the exponential distribution. Note that I/O time is the sum of channel busy time and seek time, but Fig. 5 shows only channel busy time. The distribution for seek time was not obtained, due to monitor limitations.

The system is interrupt-driven, and CPU times for servicing interrupts are considered as part of the task processing times. Since a task is modeled as an alternating series of compute and I/O times, we assume that the effect of interrupt processing is elongation of mean task compute time. The overhead for task switching can be similarly modeled.

Under normal environments the UCSB on-line system is active concurrently with batch processing and is resident in bulk core. The on-line system runs as a separate task under OS/360. It is compute-bound in nature and does very little I/O to secondary storage. Terminal-user requests are serviced with CPU time acquired periodically from OS. Thus, the effect of the on-line system on batch processing is modeled as the elongation of compute times for batch tasks, which has the same effect as interrupt processing and task switching overhead.

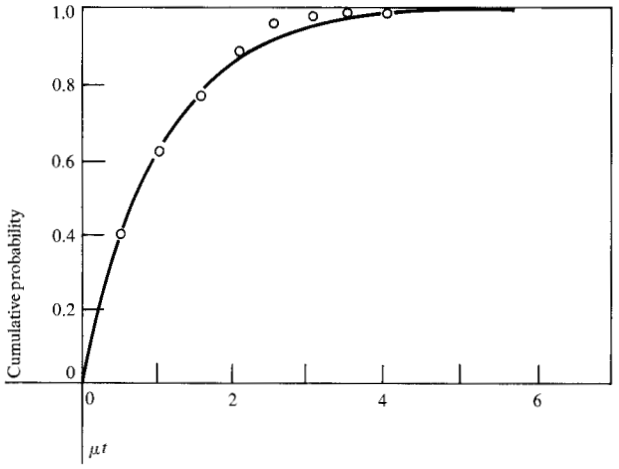

Figure 4 Empirical data of compute time under heuristic CPU scheduling approximated by an exponential distribution.

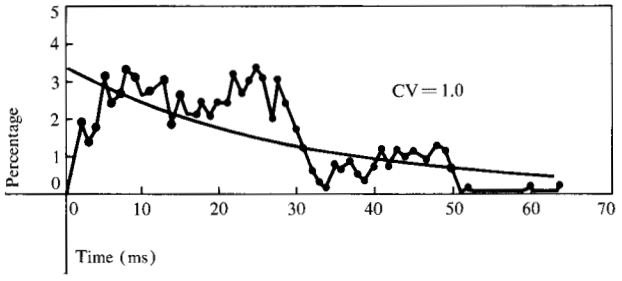

Figure 5 Empirical data of channel busy time approximated by an exponential distribution.

In view of the system structure and behavior discussed, a cyclic queue model for the System/360–75 at UCSB thus consists of one CPU and two I/O processors. The queue disciplines are FCFS for both the CPU and the I/O processors. Exponential distribution is assumed for both compute and I/O times with means $1/\mu$ and $1/\lambda$, respectively. Figure 2 is a diagram of the model structure; the performance measures can be computed from equations (3), (4), and (5). The model validation process that follows tests the sensitivity to the accuracy of the assumptions.

Model validation

We can validate the model developed in the last section by correlating observed performance with that predicted by the model with estimated values for μ, λ and N under various operating conditions. Controlled experiments are conducted with fifty sample jobs selected from our daily workload; first, effective CPU processing speed is varied, then the degree of multiprogramming is changed. These parameters are varied systematically because cause-and-effect relationships can be isolated under reproducible environments. Data obtained under normal environments are also employed for model validation.

The measurement tool is the Tesdata System Utilization Monitor, a hardware monitor. The standard IBM System Management Facility (SMF) is employed to estimate the average degree of multiprogramming, N.

We disabled the on-line system when we conducted the controlled experiments because reproducible environments cannot be easily obtained while the on-line system is active.

• *Controlled experiments*

Three sets of experiments were conducted with a controlled experiment. The first involved studies before and after replacement of the bulk core, the second varied the degree of multiprogramming, and the third examined the job and CPU scheduling processes.

Bulk core replacement During the period of this study, the eight-microsecond bulk core (LCS) that we originally had was replaced by a 1.8-microsecond bulk core (ECM). Controlled experiments, with the sample jobs as the drive load, were performed before and after the replacement. Because part of the operating system resides in bulk core, the core replacement would affect job execution times, even though batch jobs can request storage only from high-speed core (HSC).

Before the experiment, main memory is cleared of any jobs presently executing. The on-line system is disabled. The probes from the hardware monitor are attached to the appropriate signal points. The sample jobs are stacked together at the card reader so that they can be read into the system continuously. Apart from the initial and terminal transient periods, the heavy loading assumption is satisfied throughout the duration of the experiment. HASP selects jobs from the input queue based on the amounts of estimated resource requirement; the job with the smallest requirement is chosen first. Due to the job-selection algorithm and the heavy loading condition, the order of stacking the jobs at the card reader has little effect on the outcome of the experiments. The monitor is started as soon as the first job card is read, and the monitor is stopped when the last job finishes execution. Summary data are tabulated in Table 1.

Elapsed time is in seconds and represents the time between the first job card read and the last job's termination. Percentages are based on elapsed time. As one would expect, both CPU busy time and elapsed time have decreased after the replacement. CPU busy time may be divided into LCS busy time and non-LCS busy time. Knowing that the cycle-time ratio of LCS to ECM

is 8/1.8 or 4.4, we can estimate CPU busy time with ECM as

$$(1746 - 691) + (691/4.4) = 1212 \text{ seconds.}$$

This is very close to the busy time actually measured (1225 seconds).

The channel busy times do not include seek times (arm movement times). They are rather evenly spread over the two channels, especially for the run with ECM. Note that the total channel busy times are extremely close for the two runs. This proves that the environment is reproducible, and that the cause-and-effect relationship can be isolated. Total number of I/Os is the number of explicit I/O requests to the disk units (half the number of start-I/O commands). This number was not measured for the case with LCS but we can use the same value measured for the case with ECM.

The parameter $1/\mu$ for the model is estimated by

$$\frac{1}{\mu} = \frac{\text{CPU busy time}}{\text{number of I/Os}}.$$

Similarly, $1/\lambda$ is given by

$$\frac{1}{\lambda} = \frac{\text{channel busy times}}{\text{number of I/Os}} + \text{average seek time.}$$

Average seek time is average arm movement time, which is obtained via a separate experiment because we do not have enough hardware counters during one experiment. The average degree of multiprogramming, N, is estimated from information gathered by SMF. Even though five initiators are assigned to process jobs, memory limitations hold N to about 2.2.

The throughput increase for this set of jobs after replacement is 19 percent. Throughput is defined in the previous section as the product of CPU utilization and μ.

With the model parameter values we can compare the performance predicted with that measured. Figure 6 diagrams CPU utilization, p, versus ρ obtained from the model, as well as the measured performance values. Note that the model predictions are very close to actual measurements, specifically within three percent variation.

Variation of degree of multiprogramming Next, controlled experiments were performed with the number of initiators (an initiator corresponds to a job) varied from one to five. The number of initiators depicts the maximum degree of multiprogramming allowed. The drive load is again the set of 50 sample jobs. These jobs are executed from one megabyte of ECM rather than from 348 kilobytes of HSC because we do not want to be limited by memory size. This time all jobs are read into the spooling queue on disk before the start of the experiments and no job printouts are generated. To eliminate one more variable, jobs are chosen from the job queue in a first-come-first-served basis instead of by the HASP algorithm. The monitor starts to collect data as soon as the first job is selected from the job queue, and the monitor is stopped when the last job has finished execution. Summary data are tabulated in Table 2.

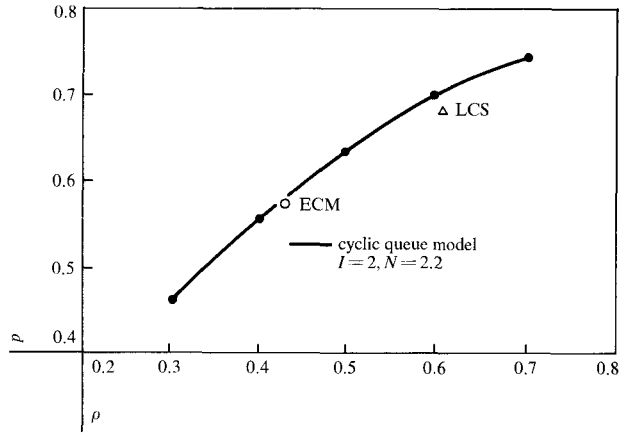

Figure 6 Model predictions and empirical data for LCS and ECM configurations.

Table 1 Controlled experiment description before and after core replacement.

	Configuration with LCS	*Configuration with ECM*
Elapsed time (s)	2580	2160
Utilization (s; %)		
CPU	1746;67.7	1225;56.7
LCS	691;27	–
Selector channel 1	670;26	834;39
Selector channel 2	928;36	760;35
Total channel (s)	1598	1594
Total no. of I/Os	60802	60802
Model parameters		
$1/\mu$ (ms)	28.7	20.1
$1/\lambda$ (ms)	26.2 + 20.5	26.2 + 20.5
$\rho = \lambda/\mu$	0.61	0.43
N	2.2	2.2
Throughput = R_c (s^{-1})	23.6	28.1

From runs one to five elapsed time decreases, but CPU busy time and total channel busy time stay about the same, which again demonstrates the consistency of these experiments. Because we lacked sufficient hardware counters, it was necessary to obtain seek (arm movement) counts and seek times by repeating these experiments. Notice that the percentage of I/O requests that need arm movements (AM) increases with each run, showing increased contention. But, for the sake of simplicity, we do not model the increase of arm movement due to increased contention. An interesting observation can be made by comparing the average seek times in Table 2 with those obtained from the last experiment (Table 1). A detailed examination shows that seek time

Table 2 Controlled experiment summary: the degree of multiprogramming is varied.

No. of initiators	*1*	*2*	*3*	*4*	*5*
Elapsed time (s)	3445	2662	2435	2278	2258
Utilization (s; %)					
CPU	2068;60	2073;78	2072;85	2076;91	2079;92
Channel 1	675;20	631;24	639;26	616;27	611;27
Channel 2	580;17	614;23	602;25	628;28	626;28
Total channel (s)	1255	1245	1241	1244	1237
Total no. of I/Os	45050	47440	47528	47749	47665
% I/O to channel 1	55.6	52.6	54.4	52.8	52.1
% I/O need AM	22.6	32	35.9	40.6	48.1
Avg. seek time (ms)	7.9	13.6	15.5	18.4	21.1
Model parameters					
$1/\mu$ (ms)	44.9	43.7	43.6	43.5	43.6
$1/\lambda$ (ms)	27.3 + 7.9	26.2 + 13.6	26.1 + 15.5	26.1 + 18.4	26.0 + 21.1
$\rho = \lambda/\mu$	1.276	1.098	1.048	0.978	0.926
N	1	1.93	2.82	3.14	3.99

increases are due mainly to the contention of the initiators for the system disk packs. Thus, even though $N = 2.2$ for the experiment in HSC, its average seek time is approximately the same as that of $N = 3.99$ in Table 2; note that both have five initiators in the system.

The model parameters $1/\mu$ and $1/\lambda$ are estimated as before. For the four- and five-initiator runs the actual degrees of multiprogramming are lower than the maximum allowed. This may be due to storage fragmentation problems, but the storage utilization measures are not easily obtained.

Employing the estimated model parameters, Table 3 lists both the CSM and CQM predictions along with the measured values. Percent deviation from the measured value is also shown. In general, the CSM predicts a lower level of resource utilization due to the probability of a job waiting for a busy channel while the other channel is free [12]. Note that for the three-initiator case the measured CPU utilization is closer to that predicted by CSM. This may be due to the fact that two jobs are allocated to one channel and one job is allocated to the other because the operating system does not split the file of a job over two channels. For the five-initiator case the observed CPU utilization is between the CQM and the CSM predicted values. For the two- and four-initiator cases the measured performances are closer to that predicted by CQM. For the one-initiator case the measured value is higher than that predicted by the models. This could be due to a certain amount of concurrent activity of HASP and job execution. We have seen that a single model is insufficient to predict system performance accurately for all operating conditions; however, the predictions from the two models form the upper and lower bounds of the actual performance. The robustness of this observation may require further testing.

Effect of scheduling Additional controlled experiments have been conducted to study the effect of scheduling. The three-initiator case from Table 2 was chosen for comparison, repeated as case A in Table 4. First, the HETM (HASP Execution Task Monitor) [4], previously employed for CPU scheduling, is removed. HETM uses a heuristic approach, as discussed in the previous section. By removing HETM we should expect a drop in performance because the compute time distribution is hyperexponential (Fig. 3). Indeed, case B in Table 4 shows a drop in CPU utilization without HETM. Note that the drop is about 10.6 percent.

Finally, case A is repeated but with HASP job scheduling instead of FCFS. HASP assigns priority to jobs based on job class (management assigned), maximum CPU kill time (a number specified by the user), and maximum number of output lines (also specified by the user). All jobs in our experiment belong to the same job class; therefore HASP weights the jobs by the other two factors. The smaller the amount of request, the higher the job priority. Thus, the shortest running jobs get through first. Case C in Table 4 is the result of this run. Note that CPU utilization has dropped seven percent from case A. Also, the average degree of multiprogramming has decreased to 2.52. A more detailed analysis of the experiment reveals that short jobs are being executed together at the beginning of the experiment and this, in turn, creates a temporary bottleneck at the reader/interpreter. The reader/interpreter is an operating system function [5] which converts the job control cards into a table form that is acceptable by an initiator. This processing tends to be I/O-bound. When the processing is complete the initiator may then request memory and I/O resources from the system to start job execution. It turns out that there is only one reader/interpreter which

Table 3 Correlation of empirical data with model predictions.

No. of initiators	*N*	ρ	*CPU utilization*				
			Measured	*CQM*	Δ(%)	*CSM*	Δ(%)
1	1	1.276	0.60	0.560	−6.7	0.560	−6.7
2	1.93	1.098	0.78	0.800	+2.6	0.740	−5.1
3	2.82	1.048	0.85	0.900	+5.9	0.840	−1.2
4	3.14	0.978	0.91	0.915	+0.5	0.855	−0.5
5	3.99	0.926	0.92	0.945	+2.7	0.890	−3.3

processes jobs sequentially; therefore, when job execution times are small compared to reader/interpreter time, a bottleneck forms.

• *Normal environment*

In the normal production environment the on-line system that resides in bulk core is active. The batch jobs being executed in high-speed core are from our normal workload. Summary data recorded by the hardware monitor over several periods of operation are presented in Table 5. Elapsed time is in units of hours. Number of I/Os to channels 1 and 2 are recorded but not shown in Table 5. Session 1 was taken when we still had the bulk core LCS. Sessions 2 through 7 are regular sessions covering various operating hours after LCS was replaced by ECM. Only the on-line system is active in session 7. During session 1 the system went down twice; session 1′ is the summary data for the interval in between. Thus, during session 1′ the system is heavily loaded with a backlog of jobs. Note that CPU utilization has dropped as expected after the replacement of LCS. Channel 1 is utilized about twice as much as channel 2 (due to suboptimal data-set configuration). The multiplexor channel busy time is due mainly to tape I/Os, printers, and card readers, and I/O from the time-sharing terminals (as indicated by the multiplexor busy time of session 7). The model parameters are estimated as before. Unfortunately, we did not have enough hardware counters to measure average seek time at the same time, but the ratio π (defined as CPU time ÷ channels 1 and 2 time) instead of ρ still gives an indication of the system load. This ratio for session 1 is much higher than for other sessions, except session 7. Mean channel busy time (defined as channels 1 and 2 busy time divided by the total number of disk I/Os) is rather constant over sessions 2 through 5. Session 7 shows that the on-line system is mainly compute-bound, with very little I/O activity to the disk units. This substantiates an earlier discussion about model development.

Next, we use the data of session 1′ for model validation. We shall assume the average degree of multiprogramming to be 2.2 as indicated in Table 1. Also, average seek time is taken to be 20.5 milliseconds (ms). Thus, from the CQM with $1/\lambda = 49.9$ ms, $\rho = 0.995$, and $I = 2$, we get CPU utilization $p = 0.825$, which is within three percent of that actually measured (Table 5).

Table 4 Controlled experiment summary: CPU and job scheduling are varied.

Case	*A*	*B*	*C*
No. of initiators	3	3	3
Elapsed time (s)	2435	2732	2614
Utilization (s; %)			
CPU	2072;85	2065;76	2070;79
Channel 1	639;26	625;23	638;24
Channel 2	602;25	580;21	601;23
Total channel	1241	1205	1239
Total no. of I/Os	47528	46098	*
% I/O to channel 1	54.4	55	*
$1/\mu$ (ms)	43.6	44.8	
$1/\lambda$ (ms)	26.1 + 15.5	26.1 + 15.5	
$\rho = \lambda/\mu$	1.048	1.077	
N	2.82	2.82	2.53

*Not recorded.

Suppose the batch workload can be represented by the sample jobs. Let us estimate what the throughput increase would be under the same conditions with ECM. The 49.7 ms of $1/\mu$ for session 1′ can be broken up into 28.7 ms + 21 ms, where 28.7 ms is the value for $1/\mu$ of the controlled experiment run with the sample jobs (Table 1) and, therefore, 21 ms is due to the on-line system. After the replacement $1/\mu$ becomes 24.9 ms, i.e., 20.1 ms + 4.8 ms, where 20.1 ms is the value of $1/\mu$ for the ECM configuration shown in Table 1, and $4.8 = 21/4.4$ (note that the cycle time ratio of LCS to ECM is 4.4). This value of $1/\mu$ is also approximately equal to the average computed from sessions 2, 4, 5, and 6. Assuming that $1/\lambda = 49.9$ ms would not change, we have $\rho = 0.5$. Then, with $N = 2.2$, the CQM predicts $p = 0.625$. Hence, the throughput increase after replacing LCS with ECM, under the same heavy loading conditions, is

$$\frac{0.625/24.9 - 0.825/49.7}{0.825/49.7} \times 100 = 51\%,$$

which is a substantial amount. Of course, this is a predicted increase and would be difficult to verify since we cannot find a normal session with exactly the same load-

Table 5 Summary profiles under normal environment.

Session	*1*	*1'*	*2*	*3*	*4*	*5*	*6*	*7*
Elapsed time	4.45	2.0	4.45	4.38	4.43	4.01	4.39	.50
Utiliz. of CPU (%)	72	80	39	33	34	45	43	34
Channel 1	30	31	28	42	28	39	39	0.6
Channel 2	15	17	12	13	13	14	14	3.4
MPX channel	12	12	13	7	21	8	8	7
$1/\mu$ (ms)	47.6	49.7	26.5	18.0	24.5	24.9	24.3	145.0
Mean channel (ms)	29.4	29.4	27.2	30.6	29.4	29.7	30.0	34.5
π = CPU/chan 1 & 2	1.62	1.69	0.98	0.59	0.83	0.84	0.82	8.39
$R_c = p\mu$	15.1	16.1	14.7	18.3	13.9	18.1	17.7	2.3

ing conditions (both batch and on-line) as those before the core replacement. Hence, one must be extremely careful when quantifying the benefits of a configuration change. This is particularly true when, as for the on-line system, the demand for service is elastic and increases as performance increases. From a qualitative standpoint, however, there is no question that both batch and on-line performance increased significantly with the core replacement.

Possible reconfigurations

The purpose of reconfiguration is the improvement of performance, which is the prime concern during heavy loading periods. There are a number of ways to do this. First, reconfigure by adding or replacing part of the equipment; second, rearrange the present configuration. Both cases are examined.

The first approach is to increase the investment in the system. In a cost study for our system [15] the indications from optimization of the present configuration are to add some additional high-speed memory. Figure 7 diagrams CPU utilization, p, versus N, the degree of multiprogramming, for both CQM and CSM predictions at $I = 2$ and $\rho = 0.5$. With an additional 512 kilobytes of high-speed core for batch jobs we can increase the degree of multiprogramming during heavy loading periods from the current operating point of 2.2 to about 4. Thus, a throughput increase of 24.8 percent could be expected, according to the CQM. Even if the behavior at $N = 4$ is closer to the CSM, we still get an increase of 6.4 percent. It seems that increasing memory size is a viable alternative to upgrade our installation.

For the second case, we examine a proposal to run the batch jobs in ECM and the on-line system in HSC. The batch jobs would have 1000 kilobytes of memory with an average degree of multiprogramming of four during heavy loading periods (as indicated in Table 2). From the last section we know that an average $1/\mu$ is 24.9 ms, which is broken up into 20.1 ms of batch job time and 4.8 ms of on-line time, and the throughput rate $R_c = p\mu = 25.1$. When batch jobs execute out of ECM their CPU time roughly doubles (note that $1.8/0.75 = 2.4$ with HSC cycle time = 0.75 microsecond), i.e., 20.1 ms becomes 40.2 ms. Similarly, on-line time is halved. Thus, $1/\mu = 42.6$ ms and $\rho = 42.6/49.9 = 0.85$, where again $1/\lambda = 49.9$. For $N = 4$, CSM predicts $p = 0.87$; therefore $R_c = 20.4$ which is less than 25.1, the original configuration. Even for CQM, the predicted value of $p = 0.93$, i.e., $R_c = 21.8$, which is also lower than before. Hence, we conclude that the proposal will not improve batch performance. The above analysis assumes that on-line load does not change with any configuration change, but in reality this assumption may not hold.

In addition, several changes have already been made or will be implemented as a direct result of this study.

First, observe that the measurements indicated that channel 1 is twice as busy as channel 2. The channel loads have been balanced by swapping system-residence disk packs, because the pack in channel 1 is much more utilized than its counterpart in channel 2.

Second, knowing that increasing the amount of memory available to batch jobs will increase throughput, we moved part of the operating system's data area to ECM and increased batch memory from 348 to 424 kilobytes of high-speed core. Controlled experiments conducted before and after the change show that CPU busy time increased slightly (about six percent), but the degree of multiprogramming has increased by an average of 0.7 job, which is enough to produce a net gain of 10 percent in throughput. Thus, this change has been implemented for the normal production environment.

Third, as the study of job scheduling in section 3 shows, resources may not be fully utilized during periods of heavy loading: Many short-running jobs could be submitted during these periods, and such jobs are selected first for execution by HASP. Student jobs are usually short and they are given highest priority for three of the five initiators normally active. Table 4 shows that the bottleneck at the reader/interpreter can cause CPU utilization to drop by seven percent; therefore, an algorithm that does not give as much perference to short jobs should be considered. The tradeoff, of course, is possible

degradation of student job turnaround time. A study is being carried out to quantify this tradeoff.

Summary

In this paper, a cyclic queue model is developed for an IBM System/360–75 with OS/MVT in the UCSB on-line system. Model validation is attempted in both a controlled batch environment and normal operating conditions. Despite obvious violations of some model assumptions and the simplicity of the model, we found reasonable correlation between the measured performance and the model predictions. The most success is demonstrated by its prediction of performance improvements from system reconfigurations; actual results show excellent agreement. The ability of the central server model [12] to predict performance is also examined. It is found to give a closer prediction of empirical performance only under certain conditions. However, both models together give the bounds for the empirical performance observed.

Under normal operating conditions the on-line system is active and is treated implicitly as slowing down the CPU (or elongating compute time of batch jobs). Statistics from a period of heavy loading have shown the model predictions to be reasonable.

Several proposals for system reconfiguration are studied by exercising the model; as a result, appropriate actions have been taken. The new understanding of the system has led to improvements in both operation and performance. Though the model is developed for a specific system, its application to other systems may also be appropriate. Its use for virtual memory systems is currently under study. Thus, the integration of empirical measurements with analytic models is a viable approach to computer performance analysis.

Acknowledgment

The authors express their gratitude for the support received from the UCSB computer center.

References and note

1. U. Grenander and R. Tsao, *Quantitative Methods for Evaluating Computer System Performance: A Review and Proposals, Statistical Computer Performance Evaluation*, Academic Press, Inc., New York, 1972.
2. H. Lucas, "Performance Evaluation and Monitoring," *Computing Surveys* **3,** 79 (1971).
3. H. Kobayashi, "Some Recent Progress in Analytic Studies of System Performance," *Proceedings of the First USA-Japan Computer Conference*, Tokyo, Japan, 1972, p. 130.
4. HASP. Houston Automatic Spooling Priority System-II (Version 3), IBM Type III Program Documentation, Order No. 360D-05.1.014, IBM Corp., Hawthorne, NY (1969).
5. IBM System/360 Operating System: MVT Job Management, Program Logic Manual. Order No. GY28-6660-9, IBM Corp., White Plains, NY.
6. S. Fuller, T. Price and N. Wilhelm, "Measurement and Analysis of a Multiprogrammed Computer System," presented at the IEEE Workshop on Performance Evaluation, Argonne National Laboratory, 1971.
7. E. Koenigsberg, "Finite Queues and Cyclic Queues," *Oper. Res.* **8,** 246 (1960).
8. W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 1, Third Ed., John Wiley and Sons, Inc., New York, 1968.
9. D. Cox and W. Smith, *Queues*, Chapman and Hall, London, 1961.
10. J. R. Jackson, "Jobshop-Like Queueing Systems," *Management Science* **10,** 131 (1963).
11. W. Gordon and G. Newell, "Closed Queuing Systems with Exponential Servers," *Oper. Res.* **15,** 254 (1967).
12. J. Buzen, "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *Comm. ACM* **16,** 527 (1973).
13. D. Gaver, "Probability Models for Multiprogramming Computer Systems," *J. ACM* **14,** 423 (1967).
14. P. Morse, *Queues, Inventories and Maintenance*, John Wiley and Sons, Inc., New York, 1958.
15. W. Chiu, "Analysis and Applications of Probabilistic Models of Multiprogrammed Computer Systems," Ph.D. Thesis and CSL Report 31, University of California, Santa Barbara, CA, 1973.
16. J. Strauss, "An Analytic Model of the HASP Execution Task Monitor," *Proceedings of the First SIGME Symposium on Measurements and Evaluations*, Palo Alto, CA, 1973.
17. K. Ryder, "A Heuristic Approach to Task Dispatching," *IBM Syst. J.* **8,** 189 (1970).
18. B. Marshall, "Dynamic Calculation of Dispatching Priorities under OS/360 MVT," *Datamation*, 93 (August 1969).

Received June 6, 1974; revised December 30, 1974

W. Chiu is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598. D. Dumont and R. Wood are located at the Department of Electrical Engineering and Computer Science, University of California, Santa Barbara, CA 92706.

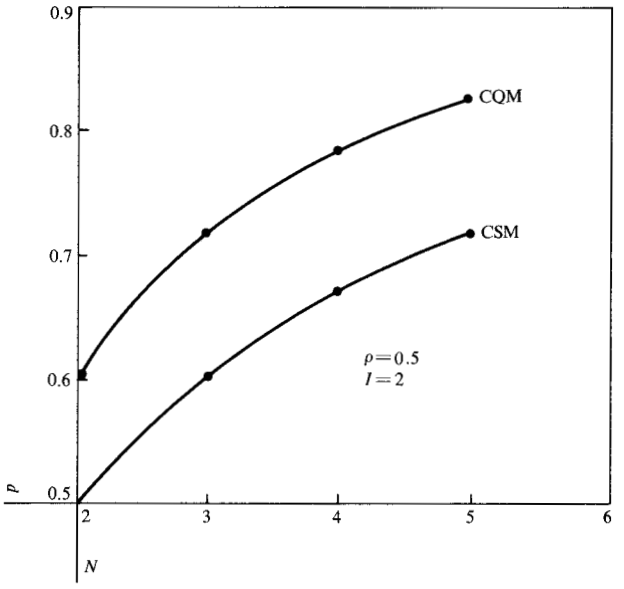

Figure 7 CQM and CSM model results for $\rho = 0.5$ and $I = 2$.