A MODEL OF PREDICTIVE CPU SCHEDULERS OF KNOWN UNCERTAINTY

by

Gene Lyonell Lasseter

TR-8

December 1972

This paper constituted in part the author's thesis for the M.A. degree at the University of Texas at Austin, December 1972.

This work was supported in part ty National Science Foundation Grant GJ-1084.

Technical Report No. 8

Department of Computer Sciences

The University of Texas at Austin

Austin, Texas 78712

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I. INTRODUCTION

Modern day computing systems, comprised of complex networks of computing equipment, have effectively moved beyond the realm of logical human rationale. Direct insight into optimal arrangement and management of the devices by the operating system is impractical and almost certainly inaccurate. "Counterintuitive" is an adjective often used in the description of proposals for the improvement of the efficiency of computing systems. Experimentation with types of system configurations is costly, often impractical, and impossible for idealized or theoretical systems. Mathematical models of computing systems, a cheaper and more adaptable method, have been used in many cases to attain the desired insight. They are useful in the study of numerous system characteristics and offer a quite accurate reflection of the considerable variability and randomness seen in the operation of most systems. However, statistical and probability models are attempted only on systems of extreme simplicity. Most real configurations are of such complexity as to render the mathematical models untenable -- being undevisable, unsolvable, or quite unattractive in solution. Simulation models, as employed in other fields of research, have been proposed for analysis of computing systems in these more complex circumstances. These models actually simulate, in a step by step manner, the action incurred by the computing systems. A simulated system can be subtly changed (in simulation) to reflect the performance of various configurations, were they to be implemented in a real system. A simulation model can be of much greater detail than a mathematical model, though the detail is obtained by an increase of the cost involved, a reduction in generality and adaptability, and is marked by uncertainty of statistical sampling accuracy. For these reasons, simulation and mathematical models should be used to supplement and enhance each other. An analytic model can be used in verification of a simulation and for insight into the effects of system modifications that are beyond the adaptive capabilities of the specific simulation.

The simulation model described in this paper is designed to determine the effects in a batch-processing, multiprogramming environment of predictive CPU scheduling, as uncertainty of the predictions varies. It has been widely postulated that the optimal CPU scheduling method would be the one which assigned the CPU to the job which would compute for the least amount of time before it relinquished the CPU to await the fulfillment of an I/O request. The basic assumption of this postulate is that I/O processing cannot keep up with CPU processing, thus forcing the CPU into periods of idleness. Handling the CPU in the manner which would most quickly funnel a job into its I/O processing phase maximizes the overlap of the CPU and I/O devices. Attainment of device overlap is the guiding principle encouraging multiprogramming environments. By the same argument, the worst scheduling tactic would be the assignment of the CPU to the job which would retain the CPU for the longest period. All other means of scheduling the CPU should, in terms of efficiency, fall between these two. Therefore, if a method could be devised which could accurately predict, by whatever means the prediction could be realized, the amount of CP time a job would consume before an I/O request halted processing, then the CPU could be scheduled to maximum advantage. Such an exact prediction seems beyond all possibility of attainment. However, predictive schemes offering less than total exactness are available. An interesting question is how detrimental to CPU efficiency is the level of uncertainty associated with a particular prediction scheme. If one develops a predictive method of known uncertainty for use in scheduling, what will be the decrease in CPU efficiency from the theoretic best scheduler? Will this predictive scheduler be of sufficient value to indicate its use in preference to other "practical" schedulers? These are questions this paper will endeavor to address.

Chapter II contains a general description of computing systems, operating systems, and the simulation of these systems. Previous work in the modeling of computing systems and the development of CPU scheduling disciplines is related in Chapter III. Chapter IV describes the system to be simulated and the means used in its simulation. Results and conclusions of the research are detailed in Chapter V.

CHAPTER II. DESCRIPTION OF COMPUTING SYSTEMS AND SIMULATIONS

A computing system is a network of computing facilities managed by an operating system. Programs, or jobs, submitted to the system request from the operating system certain facilities. When the facilities are granted the processes required by the job are performed by these facilities (not precluding either further facility requests or non-fulfillment of the processes due to conflicting or poorly defined requests). The facilities involved are memory space, central processing units, and input/output type devices. Other facilities might also be involved -- such as channels connecting memory to the I/O units -- but the simplest practical machine must include the three types of resources mentioned. At the time a resource becomes available (or is made available), the operating system determines if there are other requests for its services. If so (and, for instance in the case of memory requests, if the resource can accommodate the request), the resource is assigned to the requesting job. Where there is more than one request for the resource the resource is assigned to that job with the highest priority for the facility. Priorities are established by the rules of the operating system. It is in this manner that the orderly flow of jobs through the computing network is accomplished.

The level of efficiency attained by this flow is not readily apparent in even a most simple system to even an intense observer. Nor is it evident how slight changes in the system will affect the system's performance. Simulation models attempt to relate the efficiency of a computing system to various standards of measurement. Among the many evaluation standards employed are the following.

Throughput refers to the number of jobs which enter the system and process to completion within a specific time interval. Due to the diverse nature of most job streams, this simple count of jobs leaving the system, inherently favorable to jobs with minimum resource demands, does not present a true picture of the overall performance level of a computer system. Though if it is fully specified and used over an extended time period, throughput can serve as a rough measure of the performance level.

CPU Utilization (or Efficiency) is the percentage of the total system active time that the central processor is active.

I/O Utilization, in reference to a particular I/O unit, is the percentage of elapsed system time that the I/O unit is active. This measure is also closely related to the particular job stream. It would be most profitably used in the attempt to maintain balance of utilization levels between identical types of equipment.

Central Memory Utilization is the average percentage of central memory occupied by active jobs.

Turnaround Time refers to the elapsed time from submission of a job until its completion. Depending upon the particular application, the performance goal associated with this measure may be either to minimize the mean turnaround time or to minimize the maximum turnaround time.

These by no means represent a complete list of evaluation standards but do cover the measures in general usage. Most computer systems have been found to be "I/O-bound" -- excluding memory problems. The rate of I/O devices is considerably slower than that of a CPU, thus restricting CPU utilization. As stated earlier, the ability to keep I/O devices operating results in greater device overlap, and this ability is most sensitive to the method of CPU scheduling. For this reason and because of its profound effect on all other measures, CPU utilization is generally considered most reflective of the efficiency levels of systems under investigation.

With the bases for scoring computing system performance established, the means of simulating it will be formulated. In abstract terms a simple computing system may be viewed as a collection of queues combined with some form of priority assignment to the members of those queues. (Fig.1) The queues are composed of jobs (inputs to the system), which are individually characterized as to the length of time they wish to remain at the head of a queue and which of the other queues, if any, they desire to transfer to when they voluntarily relinquish their membership in their present queue. External to this sytem is another queue of jobs desiring to enter the system (i.e. central memory). There might in addition be specific rules, peculiar to the system at hand, governing various aspects of the queues. To define and thus simulate a computing system at this level, one has to designate the appropriate number (and identification) of queues, establish rules for handling the queues, supply fully characterized job streams, and institute a means of handling the queueing of jobs to enter the system.

The Queuing System

Within the computer system there exists one queue for each CPU and one queue for each I/O unit. By implication a job possesses (or is active on) one of the units whenever the job is at the head of that unit's queue. A record is maintained at the times at which jobs active on a device will relinquish it. Any other points in time which might affect the

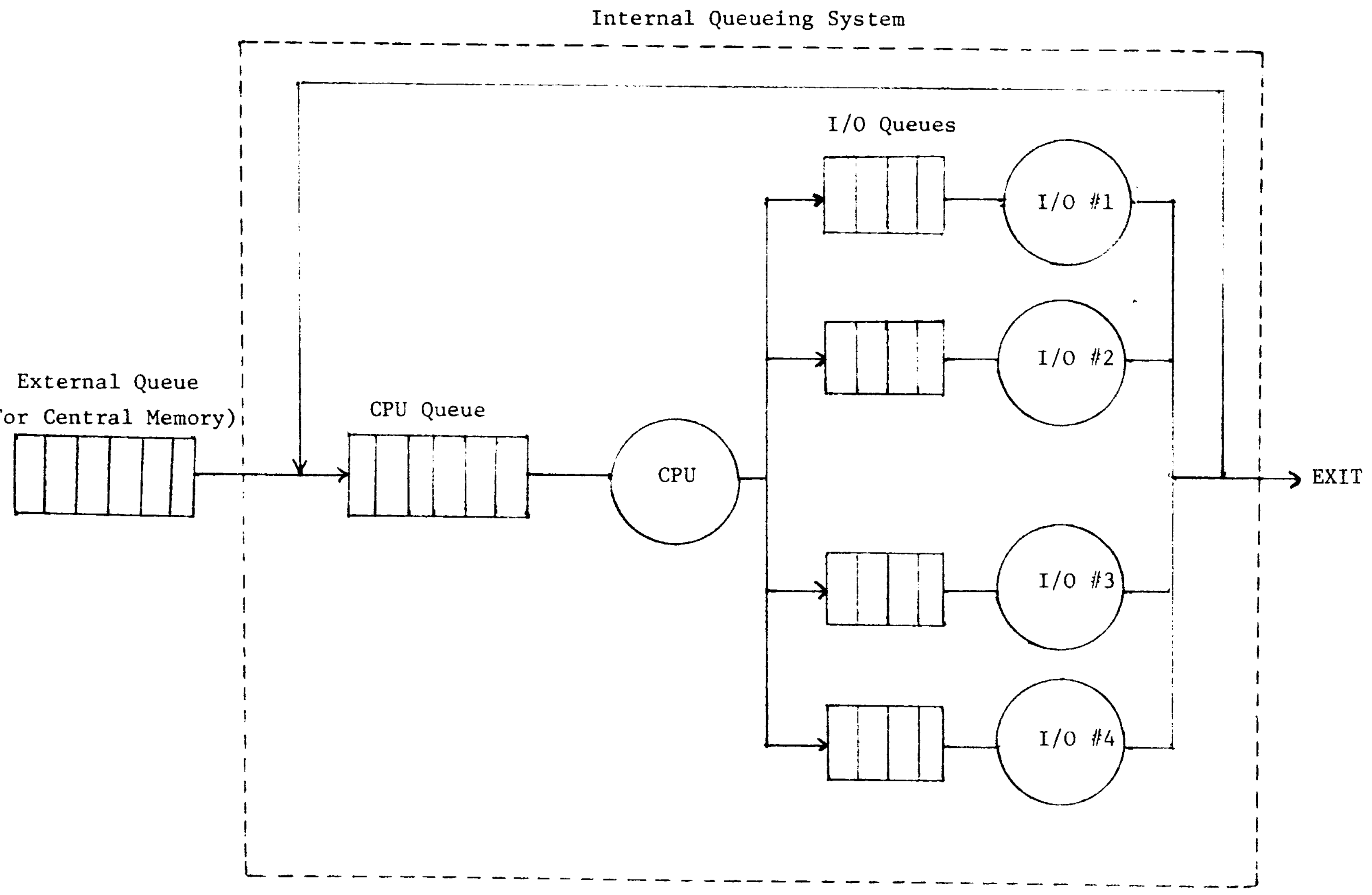


Figure 1. Queueing Model of a Simple Computer System

queues are included in the record. In this manner, referred to as event-ordered simulation, "event-step" updates of the queues are accomplished, the earliest event and its repercussions (as determined by job and operating system characteristics) being considered before later events. These repercussions include in some cases changing or cancelling later events, as evidenced by queue realignment.

The Operating System

The operating system in a multiprogramming environment handles the jobs' requests for computing facilities, deciding among any conflicting requests which jobs will be allotted the disputed facilities. Basically it is in the manner used to resolve these conflicts that operating systems are characterized. The first task of an operating system is to determine those jobs to be assigned to central memory from among those jobs in the "external queue." Next the operating system determines which of the central memory resident jobs that have requested the CPU shall be given its services. When a job active on the CPU must relinquish the CPU in order to await fulfillment of an I/O request, the operating system handles the reassignment of the CPU, determines the type of I/O service required, and attempts to supply the proper facility's services. If there are conflicting requests for that facility then the operating system assigns the device in accordance with its algorithms to one of the requesting jobs and maintains a queue for those rejected for immediate servicing. At the completion of the I/O servicing, the impediment to further central processing is removed and a request for the CPU is filed with the operating system.

The scheduling decisions referred to above are peculiar to a particular operating system. Each of the scheduling stages has associated with it numerous possible strategies. Scheduling for central memory is an important and interesting topic but for the purposes of this paper it has been ignored. Scheduling of the I/O devices is a somewhat less robust topic due to hardware constraints and in this paper, as in many real systems all I/O units are assigned to the jobs in the order in which the requests are received. Incorporated in this paper are some of the numerous schemes for the scheduling of the CPU among the jobs awaiting its services. These methods are of three general types. The disciplines may be: non-preemptive, which implies that a job only voluntarily relinquishes the CPU once it is assigned; preemptive-resume, the CPU may be seized by another job from the currently processing job for a period of time before it is returned to the seizure point and processing resumes; or preemptive-restart, the CPU, if preempted from one job by another, must start from the job's original request when control is returned. As this last is patently inefficient and thus to be avoided, the term preemption will henceforth refer to preemptive-resume type action. Detailed discussions of some of the more important scheduling disciplines, most of which can appear in both preemptive and non-preemptive forms, follow.

Shortest Burst First (SBF). The CPU is assigned to the job in the CPU queue which will process for the least time before it must halt in order to await completion of an I/O process. This is in theory the best scheduling strategy, as the jobs which have had their short bursts processed first can initiate their I/O cycle while the longer CP bursts are being processed. The CPU-I/O overlap is maximized, decreasing or or eliminating CPU inactivity incurred when all resident jobs are awaiting completion of I/O requests.

Longest Burst First (LBF). This strategy, assignment of the CPU to the job in the queue with the longest CPU burst, is assumed to be the worst possible scheduling method. Overlap of CPU-I/O activity is reduced to a minimum through the queuing of jobs in the CPU queue behind a job that will hold it for a long period. The situation created is thus more conducive to a total system cycle of CPU processing and then I/O processing than to an overlap circumstance.

Round-Robin (RR). A pure round-robin system assigns each job an identical quantum of CPU service time whenever it enters the CPU queue. On a first-come, first-serve basis each job is granted the CPU for that amount of time. If at the end of its time quantum a job has not completed its burst, the job is moved to the end of the queue, assigned an additional quantum, and must await further processing. Control of the CPU passes to the next job in the queue and the process continues. As the assigned quantum becomes arbitrarily small, the discipline becomes known as processor-sharing, with each job in the CPU queue progressing toward completion of its burst at identical rates.

First-Come, First-Serve (FCFS). FCFS is a form of RR scheduling with an infinite quantum. That job in the CPU queue with the earliest arrival time is assigned the CPU. The CPU remains in its possession until completion of its burst, at which time the CPU is switched to the job which arrived next in the queue.

Last-Come, First-Serve (LCFS). This strategy assigns the CPU to the last job to enter the CPU queue. LCFS is generally considered only in its preemptive form, with a job newly entering the CPU queue immediately seizing control of the CPU. A job with a long burst is thus more likely to be preempted than a job with a short burst. This results in a closer approximation of SBF than is obtainable with the more truly random FCFS discipline.

Predictive Shortest Burst First (PSBF). The SBF strategy as detailed above is of no practical use in real systems as the operating system cannot know in advance the length of every service time. However, the operating system can attempt to predict their lengths. All of the predictive schemes rely on a review of a job's history to determine an expected burst length or to assign a priority measure to a job. Some methods of burst length prediction are:

Complete history (17). This method predicts that the length of the next CPU service time will be equal to the mean of all previous service times for that job. The formula employed is:

$$P_n = (X_{n-1} + P_{n-1} *(n-1))/n,$$

where x_i is the i-th burst, p_i is the i-th predicted burst, and $p_1 = 0$.

Exponential smoothing (20). Using this type predictor allows more heavy weighting of the most recent past. Both the most immediate past service and predicted service time are again employed but in "exponential smoothing" they may be weighted in the manner thought most beneficial. The formula for deriving the prediction of the n-th burst is:

$$P_n = \alpha X_{n-1} + (1 - \alpha)P_{n-1},$$

with x_i and p_i as defined above and α a real number between zero and one. By larger values of α, the more recent past of a job is more heavily weighted than earlier bursts. For $\alpha = 1$, the predicted next burst has exactly the same value as the most recent actual burst. The derivation of the name of this method·is related to its general formula, which is:

$$P_n = \alpha \sum_{i=0}^{n-1} (1 - \alpha)^i X_{n-i-1} .$$

Other prediction schemes have been suggested (including the derivation of priority values from CPU-I/O burst ratios (13) or from discovery of long-short burst patterns [2]) but the above have received the most attention in the literature. A modification of these predictive schemes has been suggested to lessen the detrimental effect caused by particularly inaccurate predictions inducing the operating system to allow a job requiring a lengthy service time to monopolize the CPU for long periods. The modification involves placing an upper bound on the amount of time a job can continuously possess the CPU. If a service time exceeds that bound, the CPU is removed from the offending job. Its service requirement is repredicted (possibly using the overflowed bound time as its most recent burst) or some penalty is applied to that job's future requests.

Job Characteristics

Each job in the computing system has individual requirements for CPU time, type of I/O service, and I/O time. A trace-driven model (such as 16) employs known features of specific jobs to simulate realistic job streams. These allow extremely accurate representation of the actions of real computing systems. However, the required data is difficult and expensive to obtain and is probably of finer detail than needed for many simulations. Most simulations use job characteristics drawn from probability distributions. These probability distributions, defined by the mean and type of function, are discovered for each characteristic by analysis of experimental samplings of real systems. Various types of distributions may be applicable. Some common probability distribtuions are discussed below.

Exponential Distribution. This distribution has the characteristic that the probability of an event's occurrence in a small time interval is constant and this probability is statistically independent of the passage of time. A random variable X is said to have an exponential distribution if its density function is defined as:

$$f\ (X) = \alpha e^{-\alpha X}$$

for $\alpha>0$ and a non-negative uniformly distributed random number x. The cumulative distribution function of X is:

$$F\ (X) = \int_0^x \alpha e^{-\alpha t}\, dt = 1 - e^{-\alpha x} \quad .$$

The mean value M of X is given by:

$$M = \int_0^\infty X\ \alpha e^{-\alpha x}\, dx = 1/\alpha \quad .$$

Now F(x) is uniformly distributed between 0 and 1. If G is the random variable described by F(x), then G(x) and 1 - G(x) are equal, due to the symmetry of the uniform distribution. Thus:

$$G(x) = 1 - G(x) = 1 - (1 - e^{-\alpha x}) = e^{-\alpha x}$$

Changing to logrithmic form:

$$\alpha x = - \log_e G^{(x)} = - \log_e r, \; 0<r<1.$$

Rearranging and substituting for α:

(i) $$X = - (\frac{1}{\alpha}) \log_e r = - M*\log_e r.$$

The exponential cumulative probability is thus completely characterized by one parameter, its mean. Sampling from this distribution requires only a random value r, uniformly distributed between zero and one, for use in equation (i). The standard deviation of the distribution is equal to the mean.

Hyperexponential Distribution. The hyperexponential distribution is similar to the exponential distribution but produces random variables more likely to have "extreme" values. More of both very small and very large random variables appear in the distribution, causing its standard deviation to be larger than its mean. The hyperexponential distribution function is represented by a combination of two or more properly weighted "exponential" distribution functions. A typical such function would be:

(ii) $$F(X) = We^{-2W\alpha X} = (1 - W)e^{-2(1-W)\alpha X}, \quad 0<W<1/2.$$

To generate random numbers from an hyperexponential distribution of this type requires two random values r_1 and r_2, each uniformly distributed between zero and one. Equation (ii) involves two distributions, one with a mean value of M/2w and the other with mean M/2(1 - w). These distributions have likelihood of being chosen of w and (1 - w), respectively.

The value of r_1 dictates this choice. Once this decision is made generating the random number mimics that of the exponential distribution. The generating equations (from which one is selected) are:

(iii) $X = - \log_e r_2 * M/2W$, (for $r_1 \leq W$),

(iv) $X = - \log_e r_2 * M/2(1 - W)$, (for $r_1 > W$).

Cumulative Frequency Distribution. Often random numbers with unusual distributions or random numbers fitting observational data are desired. A method of developing such a system is to display it as a frequency distribution, indicating the number of times the variable is to fall (or has fallen) in the different intervals. The corresponding cumulative frequency distribution can be displayed by listing the percentage of the total values in the entire range that falls at or below a specified point. For x, a discrete random variable, with $P(x = b_i) = p_i$, the following table is applicable.

X	Frequency Distribution	Cumulative Frequency
b_i	$P(X = b_i) = P_i$	$q_i = \sum_{j=1}^{i} P_j / \sum_{j=1}^{n} P_j$
b_1	P_1	q_1
b_2	P_2	q_2
.	.	.
.	.	.
.	.	.
b_n	P_n	$q_n = 1$

The denominator in the cumulative frequency column is useful only where the frequency distribution refers to number of occurrences rather than probability of occurrence.

One can then use a table lookup procedure to derive discrete random variables from the desired distribution. A uniformly distributed random variable r, $0<r<1$, is generated. The table is searched to determine the values where $q_{i-1}<r\leq q$, and x is set equal to b_i.

CHAPTER III. PREVIOUS RESEARCH ON COMPUTER SYSTEM MODELING AND CPU SCHEDULING

The modeling of computing systems has been a popular topic of research for several years. MacDougall in 1970 (11) offered a tutorial paper describing the general basis for and design of simulation models for a multiprogramming computer system. The structure of a simulator, job generation, maintenance of queues, and performance evaluation are each discussed in detail.

Hellerman and Smith (8) struck a different course with their presentation of an idealized analytic model of a simple, paging, monoprogramming computer system. Any system with possible overlap of p processors (of identical speed) can at best go p times as fast as a system with no overlap capabilities. They conjectured that if standard compute times and access and flow times for auxiliary storage are all considered in particular processor overlap configurations, then the constraint on system performance from complete overlap can be calculated. They presented formulas for this determination, geared towards discovery of optimal configurations under stated timing conditions.

A much more complex probability model was developed by Gaver (6). This model, with a constant level of multiprogramming, one CPU, and identical I/O units, investigated the manner in which throughput, or CPU efficiency, is affected by CPU and I/O device speed, type of distribution displayed by CPU bursts, and core size. To this end the model was analyzed at the various levels of standard deviation on CPU burst distribution;

number of job segments in core; number of I/O devices (including an unlimited amount); and the I/O rate. Provisions for consideration of double buffering and the intervention of CPU priorities were also indicated.

Tsao, et al. (18, 19) constructed a multi-factor experiment on paging, statistically analyzed to study the interactive effects of the various factors. The factors studied included the type of replacement algorithm (for paging), memory size, size of programs, and type of grouping of system routines. By actual experimentation the most important factors affecting the paging process were determined, along with discovery of the most useful measure for comparison of the various replacement algorithms.

CPU scheduling has been of some concern in much of the recent research in computing systems. A detailed view of the several CPU scheduling orientations was presented by Coffman and Kleinrock (4). Four basic properties were noted as characterizing CPU scheduling techniques. These are: (1) preemptive vs. non-preemptive scheduling, (2) resume vs. restart (if preemption occurs), (3) how the priority is determined, and (4) when the priority is determined. The third property was discussed at length. The authors specified the various algorithms in general use for assignment of priorities, including those deriving priorities from the "state" of the computing system and from external considerations. A further variety of CPU scheduling disciplines was described in a later paper by Kleinrock (9). In it dynamic scheduling algorithms (for a time-sharing environment) were presented as a continuum dependent on the rates at which priorities are modified.

Of more direct relevance to this paper are the relatively few papers concerned with the theory and application of predictive CPU scheduling methods. Among the first of these was Stevens (17). He hypothesized that the goal of an efficient multiprogrammed computing system would be to maximize the number of jobs doing I/O simultaneously. Furthermore, to attain this the CPU should be given to the job which will relinquish it soonest (i.e. the job which will begin I/O processing soonest). The priority of a job is calculated periodically, at system intervals, as a function of the ratio of compute (CP) time consumed to I/O time consumed. In the implementation of this predictor, the computing system experienced a substantial (18%) increase in CPU utilization, though concurrent system changes make isolation of its true effect difficult. Stevens further suggested that a superior measure might be a similar priority determination using only more recent times and thus more sensitive to current processing realities.

Wulf (20) offered propoals quite similar to those of Stevens, though in a more formal manner. He defined the priority of a job in the (i + 1)th system time interval to be P_i/C_i, with

$$P_i = W_i * P_{i-1} + T_i \ , \ 0<W_1<1,$$

$$C_i = W_2 * C_{i-1} + B_i \ , \ 0<W_2<1.$$

T_i is the CP time consumed during the ith time interval by the job in question, while B_i is similarly the I/O channel time. The weights w_1 and w_2 are arbitrarily assigned to exponentially downgrade the values previously calculated. Though other changes muddle Wulf's results, the reported improvement in CPU utilization was considerable, in the range of 20 to 25%.

Marshall (13), to fulfill his goal of placing "I/O-bound" jobs at a high priority for CPU use, considered two scheduling approaches. First he implemented a reward/penalize scheme of assigning larger/smaller time slices to a job according to whether the job requested I/O before its current time slice expired. The purpose was to give I/O-bound jobs longer slices of CP time while limiting the time a compute-bound job could monopolize the CPU. The results showed this not to be a particularly effective idea. Marshall then considered the assignment of priorities for CPU use in a manner corresponding closely to the proposals of Stevens and Wulf. The priority was based on the ratio of wait time to CP time plus wait time, a value he thought to be a reasonable measure of the I/O-boundness of a job. The priorities of all jobs in core were reassessed at fixed system intervals. In actual operation, Marshall reported performance improvements in throughput of up to 23%.

Ryder (15) instituted a similar scheme to classify jobs as either I/O- or CPU-bound. The classification was based generally on the evidence exhibited by the job's most recent burst. However, factors other than this partition were used to affect CPU scheduling and the reported system improvements cannot be traced exclusively to the predictive mechanism.

Chandy and Lo (2), in a paper principally concerned with the analysis of queuing models of computer systems, suggested prediction of CPU bursts as either long or short, dependent on burst patterns discovered during the processing of a job. Observations of burst durations on a

CDC 6600 indicated that such long/short patterns do exist. Knowledge of these patterns could lead to more efficient assignment of the CPU to jobs with the predicted short bursts.

Sherman, Baskett, and Browne (16) presented a simulation model incorporating microscopic level job stream data obtained by software probes of the system. This is an example of the previously mentioned trace-driven model. Various types of CPU schedulers were modeled, including several "exponential smoothing" type-predictors and "complete history" predictors. With each of these predictors, the percentage of correct decisions (CPU assignments to the job with the shortest burst) was consistently good -- between 72 and 78%. When an upper bound was placed on the length of time a job could continuously possess the CPU, the predictive schedulers induced quite good system efficiency levels, exceeding that of all other realistic scheduling methods.

CHAPTER IV. DESCRIPTION OF THE MODEL

The Computing System

To attain maximum reflection of the specific effects on CPU utilization of the various scheduling strategies, the model was formulated simply but with realistic parameterized data taken from a real system (UT-2). A constant level of miltiprogramming is assumed, eliminating queuing for central memory and creating the effect of a closed system. There is a single CPU and four identical I/O units. The I/O units do not have equal selection probability though each I/O reference is independent of any previous reference. The operating system handles the queuing for I/O service on a first-come, first-serve basis and the queuing for CPU service according to a predetermined CPU scheduler. CPU switch time and time for "dummy" job loading are disregarded.

Job Characterization

A job is fully characterized by: (a) total CPU service requirement, (b) a set of CP bursts, (c) a set of I/O bursts, and (d) a set of I/O device specifications corresponding to each of the I/O bursts. Parameterized mean values for total CPU time, CP bursts, and I/O bursts and their respective service time distributions are used to describe the individual jobs. When a job is "loaded" into the system, its total CPU service requirement is randomly obtained from an exponential distribution function. The usual method of sampling from this distribution is employed. Given mean AVG and a 0-1 uniformly distributed random number R, the value may be determined by equation (i) in Chapter II. As will be explained below, this produces only an approximation to the actual CPU service requirements.

The length of each CP burst is generated by sampling from an exponential distribution, as in (i), on the first series of runs and from an hyperexponential distribution on the second series. Generation of the value from an hyperexponential distribution requires a preset value w ($0<w<1/2$) to specify the standard deviation of the distribution, two 0-1 uniformly distributed random variables, the mean of the distribution, and two equations (iii) and (iv) given in Chapter II. The duration of each I/O burst is sampled from an exponential distribution as described by (i).

A cumulative frequency probability distribution is used in a table lookup procedure to specify the I/O unit to relate with each I/O burst. A 0-1 uniformly distributed random number is associated with the cumulative probability to determine the I/O unit identification number inthe following table.

I/O Unit ID	Cumulative Probability
1	0.500
2	0.667
3	0.833
4	1.000

I/O unit #1 thus has three times the probability of being selected as each of the other three units.

A job is considered to consist of a pattern of CPU-I/O bursts with an associated I/O unit for each pair of bursts. When the generation of a CP burst reduces the job's total remaining CP requirements to below the mean for CP bursts, the newly generated pair of bursts are specified as the final pair for that particular job and the total CP time expended on that job is adjusted accordingly.

CPU Scheduling Strategies

Several different schemes for scheduling the CPU were modeled. These were:

(1) SBF - both preemptive and non-preemptive forms,

(2) PSBF - preemptive with both confidence regions and confidence levels varied,

(3) FCFS,

(4) LCFS - preemptive form,

(5) RR - with quanta of 8, 16, 32, and 64 milliseconds,

(6) LBF - with a preemption quantum of 32 milliseconds.

Modeling Predictive SBF Strategies

For the purposes of this paper, the particular strategy used by the computing system to obtain a burst prediction is disregarded. The important factor is the degree of accuracy of the predictions. Thus rather than attempt to model one of the previously mentioned prediction methods, a predicted burst length is derived from the true burst by employing only the required level of accuracy for the predictions. It is assumed that, with a particular parameterized level of confidence, one can guarantee that the predicted burst lies within a certain confidence interval of the true burst. These predictions are uniformly distributed within this allotted confidence interval. The width CI of a confidence interval, given the true burst time T and the allowable percentage variation VAR, is determined by the formula:

$$CI = 2 * VAR * T.$$

To obtain the predicted burst P, a 0-1 uniformly distributed random variable R is used in the following:

$$P = T + CI * (R - 0.5).$$

Generating those (randomly determined) predicted bursts which do not conform to this interval entails a table lookup procedure. A 0-1 uniformly distributed random number is compared with the cumulative probabilities listed in the following table to attain the multiplicative factor M.

MULTIPLICATIVE FACTOR M	CUMULATIVE PROBABILITY
-4.0	0.1426
-3.0	0.2085
-2.5	0.2603
-2.0	0.3303
-1.75	0.3757
-1.50	0.4310
-1.25	0.5000
1.25	0.5690
1.50	0.6243
1.75	0.6697
2.0	0.7397
2.5	0.7915
3.0	0.8574
4.0	0.9388
6.0	0.9725
8.0	0.9876
10.00	1.0000

The multiplicative factor obtained in this tabular search is used in the following equation to produce a predicted burst outside the proscribed confidence interval.

P + T + M * CI.

If the predicted burst derived in this manner is less than zero it is set equal to zero.

Input Data

The data input to the model for the purposes of job generation, prediction generation, simulation control, and software control are the following:

(1) the tabular values required to derive the multiplicative factor for use in the burst prediction routine,

(2) the number of jobs to be generated during the course of the simulation,

(3) the fixed level of multiprogramming,

(4) the mean total CP service time,

(5) the mean CP burst time,

(6) the mean I/O burst time,

(7) the length of the quantum to be used for CPU scheduling purposes in the model,

(8) a flag indicating whether or not the scheduling scheme is predictive,

(9) the confidence level to be used for burst prediction generation,

(10) the confidence interval to be used for burst prediction generation.

System Performance Measures

Several performance measures are recorded for evaluation of the various CPU scheduling strategies. These are:

(1) the number of times the CPU is switched from one job to another,

(2) a device utilization percentage for the CPU and each I/O unit, and

(3) the number of jobs completed per simulated second (throughput).

CHAPTER V. RESULTS AND CONCLUSIONS

Tables I through IV display the results of the simulated models. The performance levels attained for the various CPU scheduling disciplines are consistent for each of the three burst distributions. CPU utilization appears to be the most responsive of the performance measures. As expected the SBF strategy produced levels of efficiency superior to that of any of the other schedulers. On only a slightly lower level were the several predictive schedulers. The CPU utilization measure for these schedulers approximately ranged from 1% to 4% below this measure for SBF scheduling, depending upon the degree of uncertainty imposed on the predictions. RR, with small quantum, and LCFS strategies produced performance levels at the lower end of this range. It is only when the predictions are substantially inaccurate that these two disciplines can match the efficiency of the predictive schedulers. FCFS scheduling falls from 1% to 4% below these, the degradation increasing as the standard deviation of the burst lengths increases. The CPU utilization for LBF, easily the worst of the strategies, forms a lower bound for the measure on these models. Its utilization factor is 10 to 12% lower than that of SBF.

Almost all previous research had shown PSBF to be a viable alternative for CPU scheduling. Sherman, Baskett, and Browne (16) in their trace-driven model had produced results consistent with these. However, they used a real system and definite predictive methods, and thus general formulations cannot be deduced. The results reported here indicate for any system the degree of accuracy needed for a predictive scheme to induce system improvements over the ordinary schedulers. This degree of

Table I: Performance Measures of the Models with Exponentially Distributed CP Bursts (σ^2 = MEAN)

Measure / Scheduling Discipline	% CPU Utilization	Throughput (Jobs/second)	CPU Switches (Thousands)
Preemptive SBF	71.8	1.41	36.2
Non-Preemptive SBF	69.8	1.39	25.6
PSBF, CL = 0.9, VAR = 25%	71.5	1.40	35.0
VAR = 75%	70.9	1.39	34.9
VAR = 100%	69.3	1.41	33.6
RR, Q = 8	68.2	1.37	99.8
Q = 16	68.6	1.39	59.2
FCFS	67.9	1.33	25.9
LBF, Q = 32	62.3	1.21	48.3

Table II: Performance Measures of the Models with Hyperexponentially Distributed CP Bursts ($\sigma^2 = 5*MEAN$)

Measure / Scheduling Discipline	% CPU Utilization	Throughput (Jobs/second)	CPU Switches (Thousands)
Preemptive SBF	71.3	1.38	75.0
PSBF, CL = 0.9, VAR = 25%	71.1	1.37	74.7
VAR = 100%	69.4	1.33	73.0
VAR = 300%	67.7	1.31	70.3
CL = 0.5, VAR = 25%	69.9	1.31	70.3
VAR = 100%	68.4	1.32	71.2
VAR = 300%	67.3	1.31	69.6
RR, Q = 8	67.5	1.27	210.2
Q = 16	67.6	1.31	213.7
LCFS	67.2	1.29	86.0
FCFS	65.7	1.29	51.4
LBF, Q = 16	61.0	1.20	139.7

Table III: Performance Measures of the Models with Hyperexponentially Distributed CP Bursts (σ^2 = 10*MEAN)

Measure / Scheduling Discipline	% CPU Utilization	Throughput (Jobs/second)	CPU Switches (Thousands)
Preemptive SBF	71.5	1.31	82.0
PSBF, CL = 0.9, VAR = 25%	70.8	1.28	80.6
VAR = 100%	68.8	1.25	77.5
VAR = 300%	67.3	1.21	76.1
RR, Q = 8	67.1	1.22	217.2
Q = 16	67.0	1.20	134.8
LCFS	66.7	1.21	91.4
FCFS	62.5	1.14	56.0
LBF, Q = 32	59.7	1.09	103.9

Table IV: Percentage CPU Utilization for PSBF Models of Differing Confidence Limits and Confidence Regions (Hyperexponentially Distributed CP Bursts, σ^2 = 5*MEAN)

CL / VAR	0.9	0.8	0.7	0.5	0.3
25%	71.1	70.7	70.5	69.9	69.8
50%	71.0	69.9	70.4	69.4	68.7
100%	69.4	70.0	68.8	68.4	68.3
200%	68.1	68.2	67.9	68.1	68.0
300%	68.4	67.9	67.6	67.3	67.6
500%	67.7	67.6	67.7	68.1	67.4

accuracy is shown to not be particularly extraordinary. This might explain the excellent results obtained by the relatively primitive prediction algorithms used in some experiments. (15, 17). The tables show that the CPU utilization measure is moderately insensitive to variations in the prediction's confidence level and confidence region. As the predictive accuracy becomes smaller the further system degradation also becomes progressively smaller (Table IV). It appears that the important factor is not the preciseness of the prediction but whether the prediction is to even a small degree related to the true burst. The previously mentioned propositions advanced by Chandy (2) are most harmonious with these results. Scheduling of the CPU according to derivation of long/short burst patterns would approximately coincide with PSBF scheduling with ample predictive uncertainty, and thus should be a beneficial discipline. Both RR and LCFS require much greater numbers of CPU switches than PSBF. For those systems which consume non-trivial amounts of system overhead in CPU switching, performance improvements associated with a PSBF scheduler would indeed be significant.

Factor Analysis

Table IV displays the percentage CPU utilization of the PSBF model for various levels of the percentage confidence region VAR and the confidence level CL. The results in the table were further analyzed to determine the relative sensitivity they exhibited to these two factors. Both linear and non-linear models were examined in a multiple regression analysis. The linear model used was:

$$F = c_1 * CL + c_2 * VAR + c_3.$$

The non-linear model was:

$$F = c_1 * CL + c_2 * VAR + c_3 + c_4 * CL * VAR.$$

F in both cases represented the measure of CPU utilization. The coefficients discovered were markedly similar in both analyses.

Analysis Results	c_1	c_2	c_3	c_4	Std. Error
Linear Model	-2.65	-0.68	71.14	-	0.412
Non-linear Model	-2.61	-0.70	71.30	-0.13	0.495

The results indicate that the relationship is more nearly linear -- indeed the standard error is larger for the non-linear model. If the coefficients c_1 and c_2 are normalized (as the variables CL and VAR have different numerical ranges), c_1 remains somewhat larger than c_2, suggesting that the model is slightly more sensitive to changes in the confidence level than to the width of the confidence region.

Derivation of the Results

In the simulation models some standard parameters were used for all runs. These were:

Level of multiprogramming	5
Mean total CP time..........................	500 milliseconds
Mean CP burst.............................	40 milliseconds

Mean I/O burst..........................100 milliseconds

Total number of jobs generated

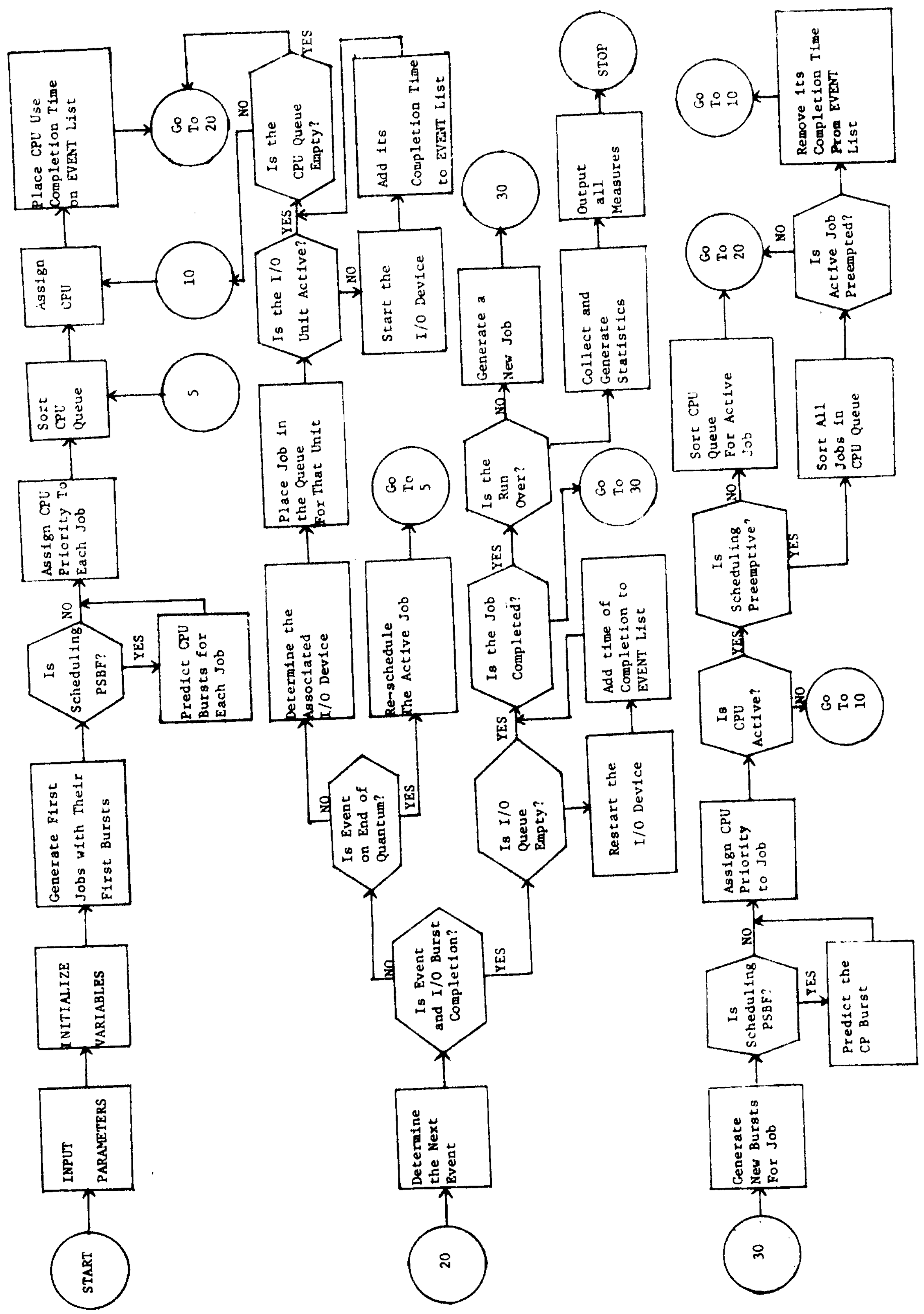
Exponentially distributed CP bursts 2000

Hyperexponentially distributed CP bursts ...4000

The results listed in the tables represent average values of three loops through the simulation. These loops are differentiated by different starting points for the random number generator and produce three completely different job streams. For hyperexponentially distributed CP bursts two series of runs were made. On the first the standard deviation of the distribution was 5*MEAN ($w = 0.2$) and on the second it was 10*MEAN ($w = 0.1$).

Reproduceability

The results of the various simulation runs seemed notably steady and the variation among different runs of the same model was slight. Probably runs encompassing a much shorter length of simulated system time would have produced results of nearly equal accuracy but the greater time was employed so that the system would be assured of approaching a steady-state balance. The simulation model was verified analytically by the ASQ system (as described in 3) for FCFS scheduling with exponentially distributed CP bursts.


START
INPUT PARAMETERS
INITIALIZE VARIABLES
Generate First Jobs with Their First Bursts
Is Scheduling PSBF?
YES
NO
Predict CPU Bursts for Each Job
Assign CPU Priority To Each Job
Sort CPU Queue
5
Assign CPU
10
Place CPU Use Completion Time on EVENT List
Go To 20
20
Determine the Next Event
Is Event and I/O Burst Completion?
NO
YES
Is Event on End of Quantum?
NO
YES
Determine the Associated I/O Device
Re-schedule The Active Job
Go To 5
Place Job in the Queue For That Unit
Is the I/O Unit Active?
YES
NO
Start the I/O Device
Add its Completion Time to EVENT List
Is the CPU Queue Empty?
YES
NO
Is I/O Queue Empty?
YES
Restart the I/O Device
Add time of Completion to EVENT List
Is the Job Completed?
YES
Go To 30
Is the Run Over?
NO
Generate a New Job
30
Collect and Generate Statistics
Output all Measures
STOP
30
Generate New Bursts For Job
Is Scheduling PSBF?
YES
NO
Predict the CP Burst
Assign CPU Priority to Job
Is CPU Active?
NO
YES
Go To 10
Is Scheduling Preemptive?
NO
YES
Sort CPU Queue For Active Job
Go To 20
Sort All Jobs in CPU Queue
Is Active Job Preempted?
NO
Go To 20
Remove its Completion Time From EVENT List
Go To 10

Generalized Flowchart of the Simulation Model

BIBLIOGRAPHY

1. Baskett, F., Mathematical Models of Computer Systems, Ph.D. Dissertation, University of Texas at Austin, 1970.

2. Chandy, K. M., "Queueing Models in Computer Systems," Proc. of the First Texas Symposium in Computer Systems, University of Texas, Austin, Texas (June 1972).

3. Chandy, K. M., T. W. Keller, and J. C. Browne, "Design Automation and Queueing Networks," Proc. Ninth Annual Design Automation Conference, 9, pp. 357-367 (June 1972).

4. Coffman, E. G., and L. Kleinrock, "Computer Scheduling Methods and Their Countermeasures," Proc. AFIPS 1968 Spring Joint Computer Conference, Vol. 32, pp. 11-21.

5. Fenichel, R. R., and A. J. Grossman, "An Analytic Model Multi-Programmed Computering," Proc. AFIPS 1969 Spring Joint Computer Conference, p 717.

6. Gaver, D. P., Jr., "Probability Models for Multiprogramming Computer Systems," Journal of the ACM, Vol. 14, No. 3, July 1967, p 423.

7. Gordon, G., System Simulation, Prentice-Hall, Englewood Cliffs, N. J., 1969.

8. Hellerman, H., and H. J. Smith, "Throughput Analysis of Some Idealized Input, Output, and Computer Overlap Configurations," Computing Surveys, Vol. 2, No. 2, June 1970, p 111.

9. Kleinrock, L., "A Continuum of Time-Sharing Scheduling Algorithms," Proc. AFIPS 1970 Spring Joint Computer Conference, Vol. 36, pp. 453-458.

10. Lan, J. C., "A Study of Job Scheduling and Its Interaction with CPU Scheduling," TSN-24, Computation Center, University of Texas at Austin (December 1971).

11. MacDougall, M. H., "Computer System Simulation: An Introduction," Computing Surveys, Vol. 2 (September 1970), pp. 191-209.

12. McKinney, J. M., "A Survey of Analytical Time-Sharing Models," Computing Surveys, Vol. 1 (June 1969).

13. Marshall, B. S., "Dynamic Calculation of Dispatching Priorities Under OS/360 MVT," Datamation (August 1969), pp. 93-97.

14. Ramamoorthy, C. V., K. M. Chandy, and M. J. Gonzales, "Optimal Scheduling Strategies in a Multiprocessing System," IEEE-TC Vol. C-21, No. 2 (February 1972).

15. Ryder, K. D., "A Heuristic Approach to Task Dispatching," IBM System Journal 8, 3 (1970), pp. 189-198.

16. Sherman, S., F. Baskett, and J. C. Browne, "Trace Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System," University of Texas at Austin, 1970.

17. Stevens, D. F., "On Overcoming High-Priority Paralysis in Multiprogramming Systems: A Case History," Comm. ACM, 11 (August 1968), pp. 539-541.

18. Tsao, R. F., L. W. Comeau, and B. H. Margolin, "A Multi-Factor Paging Experiment: I. The Experiment and the Conclusions," IBM Research Report RC 3443 (July 1971).

19. ____________, and B. H. Margolin, "A Multi-Factor Paging Experiment: II. Statistical Methodology," IBM Research Report RC 3522 (August 1971).