This is to certify that the

thesis entitled

A MODEL OF PROGRAM PERFORMANCE IN MULTIPROGRAMMING SYSTEMS

presented by

Chester Terrance Trahan

has been accepted towards fulfillment
of the requirements for

Ph.D degree in Management Science

Richard C. Henshaw, Jr.
Major professor

Date 11/6/79

O-7639

A MODEL OF PROGRAM PERFORMANCE IN MULTIPROGRAMMING SYSTEMS

By

Chester Terrance Trahan

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Management

1978

ABSTRACT

A MODEL OF PROGRAM PERFORMANCE IN MULTIPROGRAMMING SYSTEMS

By

Chester Terrance Trahan

Good planning information is a continuing necessity for Data Processing management. Along with the need for good performance information in planning to meet demand for computer services, there is a need for individual program performance prediction in computer job scheduling.

Some of the information necessary for planning is collected based on performance measurement (usually by system software). This must be augmented by benchmarks, simulation or other techniques to predict system performance under a specific set of circumstances.

The use of an analytic model has been proposed as the most flexible method of predicting computer performance for medium scale, priority-interrupt driven operating systems. To this end an analytic model was developed with cyclic queuing submodels for the Central Processing Unit and the Input/Output units and a deterministic "independent reference" submodel for program paging behavior. The model was programmed in APL [1] and Newton's Method and Aitken's Delta Square [2] algorithm were used to achieve convergence of the model to equilibrium solutions.

A set of parameterization runs were made on a System/370 Model 148 and their measurements were then used to estimate the system parameters: I/O overhead, paging overhead, Page-in/Page-out ratios, and page "weights" for the paging submodel. A "synthetic" program was written so that computer usage and paging behavior could be controlled internally and real storage, Input/Output behavior and priority could be controlled externally to the program. Several variations of the "synthetic" program were measured and their characteristics estimated.

Following parameter estimation, the estimated values were used in the APL model to predict the performance of the experimental programs in an experiment designed as a 2X2X2X2X2 half-replicate factorial. The actual experiments were then conducted and the experimental results compared to the predictions. The results of the experiment showed good prediction in the area of working set sizes and elapsed times and aggregate I/O rates.

The results showed a sizable error in page rates, channel utilization, overhead and CPU utilization. The nature of these results was attributed to the inadequacy of the independent reference model in representing paging behavior. These results reinforced Belady and Kuehner's conclusions about the unsuitability of independent references [3] as a model for program paging. Modifications were then made to the model and the predictions of the

revised paging model showed good agreement with the experimental data. The revised paging model as well as several previously developed models showed good agreement with paging behavior for programs executing with constrained memory. However, all of the models examined showed a poor fit under conditions of loose memory constraints.

BIBLIOGRAPHY

1. Iverson, K.E. A Programming Language, New York, John Wiley & Sons, Inc., 1962.

2. Isaacson, E., and Keller, H.B. Analysis of Numerical Methods, New York: John Wiley & Sons, Inc., 1966.

3. Belady, L.A., and Kuehner, C.J. "Dynamic Space-Sharing in Computer Systems", Communications of the ACM, Vol. 12,No. 5, (May 1969), pp. 282-288.

To Thelma

ACKNOWLEDGMENTS

I would like to thank the chairman of my guidance committee, and present chairman of the Management Department, Professor Phillip Carter for his encouragement. I also want to thank the members of my dissertation committee, Co-chairmen Professors Richard Henshaw and Herman Hughes, and Professor Gerald Park. I also owe a debt of gratitude to my manager, Paul Beukema, for his help in getting permission for me to perform my experimental work at IBM.

East Lansing, Michigan
1978

C.T.T.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

e(w)	The average process time between page faults.
f(.)	A function giving the difference between the average instantaneous I/O access rates on two consecutive iterations of the model.
p	A real number between zero and one. A probability.
w(j)	The average working set size for program j (pages).
A(j)	The paging activity index for program j (references/instruction).
ALP	CPU overhead for a non-paging I/O cycle (seconds).
ARAT(i)	Total access rate for all programs to device i (accesses/second).
ARAT(i,j)	The access rate for program j to device i (accesses/second).
ARAT(i,j,k)	The access rate for program j to file k on device i (accesses/second).
B(j)	The length of a CPU service interval before an I/O or paging operation is generated (seconds).
BETA(j)	The ratio of page-reads to page-writes for program j.
BETA	The system ratio of page-reads to page-writes.
BS(i,k)	The average block size (physical record size) for file k of device i (bytes).
BSP	The block size of a page in the virtual system under discussion (bytes).
C(j)	The completion time for program j (seconds).

CA	The coefficient of variation for the distribution of I/O requests for disk.
CER(j)	The instantaneous page exception rate for program j (exceptions/second of process time).
CI(i,k)	The cylinder location of the mid-point of the index area for indexed file k on device i.
CK(j)	A factor used in calculating the paging response time for program j.
CL(i,k)	The number of cylinders in the index portion of file k on device i.
COH(j)	The CPU utilization for overhead due to program j.
CP(j)	The CPU utilization due to program j not including overhead (COH(j)).
CPP(j)	The total CPU utilization due to program j including both CP(j) and COH(j).
Cs(j)	The coefficient of variation for the distribution of completion time for program j.
CS	The coefficient of variation for the distribution of disk service time.
CTIME(j)	The total CPU time attributed to program j by operating system job accounting routines. Total problem-state time (seconds).
CYL(i,k)	The size of file k on device i (cylinders).
CYT(j)	The ratio of the apparent machine instruction rate to the nominal machine instruction rate for program j.
D(j)	The amount of CPU processing time pre-empted by the channel in performing an average I/O operation for program j.
DC	The amount of CPU time "stolen" by the channel for each byte of data transferred (seconds).
DEL	The average CPU overhead for a paging cycle (seconds).
DI(i,j,k)	The probability of an access by program j to file k on device i.
DI(i,k)	The probability of an access to file k on device

i by any program during a specified interval.

DS — The amount of CPU time "stolen" by the channel to initiate an I/O operation (seconds).

ET(j) — The elapsed time per cycle for program j including initial wait W(j), completion time C(j), and I/O wait time IC(j) (seconds).

ETA(j) — The average I/O wait time during a paging cycle, ETA(j) = (1 + Beta(j))*PHI(j) (seconds).

EXP(N) — The paging expansion factor. The tendency of overall system paging rate to increase with an increase in multiprogramming set due to page frame contention.

FAC(j) — The expected fraction of completion time which a program must wait if completion of it's I/O finds the CPU busy servicing program j. If Cs(j) is the coefficient of variation for the completion time for program j, FAC(j) = (1+Cs(j))/2.

FN — The average system real-time paging rate. The sum of the individual program real-time paging rates (pages/second).

H(j) — A factor used in calculating the induced completion time due tc the completion of I/O for a higher priority program while a lower priority program is in the interrupted state.

IO(j) — The average response or wait time for any type of I/O operation by program j including paging (seconds).

K — The average CPU instruction execution rate (instructions/second).

KS — The slope of the expression for seek time. The average seek time per cylinder (seconds/cylinder).

LAM(j) — The average instantaneous I/O rate for program j. This means that while program j waits for completion of paging or I/O the operation completes with rate LAM(j) (completions/second).

LC(i,k) — The mid-point of the data area of file k cf device i (cylinder).

L(j) — The weighting factor for program j in a biased memory allocation scheme.

M(j)	The "critical memory". The smallest value of program j's working set such that program j's paging rate is less than .5 pages per second.
MU(j)	The instantaneous CPU access rate for program j in the absence of paging. MU(j) = CTIME(j)/(N(j)+1).
N(i,j,k)	The average number of non-paging I/O operations to file k of device i issued by program j during it's execution.
N(i,k)	The average number of non-paging I/O operations issued by all programs in the multiprogramming mix to file k of device i during some specified interval.
N(j)	The total number of non-paging I/O operations generated by program j during it's execution.
N	The total number of programs in the multiprogramming mix.
OHT(N)	The total system CPU overhead with N programs in the multiprogramming set.
PG(j)	The total number of cycles in program j's execution which terminate in paging exceptions. The number of pages read for program j.
PHI(j)	The average time it takes for program j to read or write a page from virtual memory (seconds).
PI(j)	The average real-time page exception rate for program j. PI(j) = CP(j)*CER(j) (exceptions/second of real time).
PO(j)	The total number of page-write operations on behalf of program j during it's execution.
PR(j)	The the sum of the page-read and page-write rates for program j. The total paging rate for program j.
RC(i,k)	The number of data records per cylinder for indexed file k cn device i.
RCU(i,j,k)	The average channel utilization caused by program j accessing file k on device i.
RCU	The average total channel utilization for some specified channel during a specified interval.
RCU(j)	The average total channel utilization that is

	apparent to program j. The expected channel utilization on the condition that program j is not using the channel.
RD	The period of rotation of a disk device. The length of time required for the platters or disks to complete one revolution (seconds).
R	The total amount of pagable memory. The total amount of real memory minus fixed memory (pages).
S(j)	The total amount of virtual memory required by program j for execution--program j's address space (pages).
SK(i,j,k)	The average seek distance experienced by program j when accessing file k of device i (cylinders).
SK(i,k)	The average seek distance for any program in the multiprogramming mix when accessing file k on device i (cylinders).
SK2(i,j,k)	The second moment of the seek distance for program j when accessing file k on device i (cylinders**2).
S1(i,j,k)	The average seek distance for program j when accessing the index area of indexed file k on device i (cylinders).
S2(i,j,k)	The average seek distance for program j when accessing the data portion of indexed file k on device i (cylinders).
T	The amount of time required to move a disk access mechanism exactly one cylinder (seconds).
T(j)	The aggregate I/O rate for all but program j. This symbol is also used to represent a delay term in calculating initial wait.
TAU(j)	The average amount of CPU usage due to program j per cycle, including the overhead for paging and/or normal I/O (seconds).
TRD(i,j,k)	The average total response time for program j accessing file k on device i, including wait time (seconds).
TS(i,j,k)	The average seek time for program j when accessing file k on device i (seconds).
TS1(i,j,k)	The average seek time for program j when access-

ing the index area of indexed file k on device i (seconds).

TS2(i,j,k) The average seek time for program j when accessing the data area of indexed file k on device i (seconds).

TSC(i,k) The average channel service time for any program accessing file k on device i (seconds).

TSC1(i,j,k) The average channel service time for program j's access to the index area of indexed file k on device i (seconds).

TSC2(i,j,k) The average channel service time for program j's access to the data area of indexed file k on device i (seconds).

TSD(i,j,k) The average device service time for program j when accessing file k on device i directly or sequentially (seconds).

TSD1(i,j,k) The average device service time for program j when accessing the index area of indexed file k on device i (seconds).

TSD2(i,j,k) The average device service time for program j when accessing the data area of indexed file k on device i (seconds).

TWC(i,j,k) The average channel wait time due to blocking by other programs experienced by program j when accessing file k on device i (seconds).

TWR(i,j,k) The average channel wait time due to the channel being busy when the record to be read or written passes under the read/write heads. This is the wait time experienced by program j when accessing file k on device i (seconds).

W(j) The initial wait time experienced by program j between the completion of an I/O or paging operation and program j's next access to the CPU (seconds).

XI(j) The average I/O response time experienced by program j for non-paging I/O to all of it's files (seconds).

INTRODUCTION

Performance Planning

Data Processing management has a continuing need for planning information. Long and short range planning is necessary so that the data processing needs of the organization can be met without undue disruption of service and unnecessary expense.

Lead times for equipment delivery schedules require that equipment be ordered several months to years before the equipment is actually needed. To assess the impact on customer service and installation stability, the data processing management team must be able to estimate the effect of changes in software or hardware.

Software changes can be to application software or systems software. Application software is the collection of programs and procedures in an installation which are written to support the business functions of the organization. Programs which do payroll, schedule manufacturing operations, and maintain inventory accounts are examples of application programs. Programs which generally support the functions of the data processing organization are called system software. Program compilers, sort routines, and even

the collection of programs which control the elements of the computer system--the operating system--are all examples of system software. Hardware changes may be made to the central processor (CPU), or to the peripheral or input/output (I/O) devices such as disk drives, tape drives, printers, drums, card equipment and teleprocessing terminals. The change may be either replacement, acceleration of operating speed, or augmentation of the capacity of the device(s) or unit(s) under consideration.

Part of the information needed by data processing management is provided by their data processing system in the form of job accounting data collected during system operation, while the systems requirements come from the business plans developed by corporate planners. The remainder of the information required by data processing management to fulfill their responsibilities is derived from industry publications, vender literature and proposals, staff research and intuition.

Information in the third category tends to be associated with performance, capacity and the capability of computer systems. Typical questions that one hears in this category might be: will the addition of that new disk drive improve throughput by 10% or are other changes required? how many programs should be running concurrently to optimize throughput? will the installation of the new CPU be sufficient to handle a projected 15% increase in workload

se

be

sh

to

ar

wh

ba

ab

ra

si

co

la

ar

be

(c

the

im

ti

the

re

(3)

pe

te

during the next twelve months? will the proposed system meet the performance specifications? how will the schedule be affected if the third shift update is moved to the first shift?

The answers to such questions are not only of interest to data processing management and systems analysts but they are of key importance to vendors of hardware and software. While system collected job accounting data can supply the basis for an analysis which may yield the answers to the above questions, it cannot provide answers.

The means of deriving answers to the above questions range from intuitive guesses to in-depth analysis and simulation. In some cases decisions are made to increase computer capacity without in-depth analysis because of a lack of analytical skills. In other instances benchmarks are performed as a means of alleviating this situation. A benchmark is the execution of an actual set of programs (currently implemented on an installed or "base" system) on the proposed or "target" system. This approach is usually impractical for the following reasons: (1) the amount of time available on a target system is sometimes limited, (2) the data files from the system being modeled cannot be removed because they are needed for continuing processing, (3) and the number of disk packs available for extended periods of time is often inadequate for a complete benchmark. Even if the foregoing problems can be surmounted, the

logistics of duplication and transportation of card decks, disk packs and tapes can create additional problems.

When one considers the conflicts involved in scheduling computer operators for the benchmark as well as continuing production work, it is easy to see why the approach generally taken is the preparation of a "representative" sample of job streams which are supposed to embody the characteristics of the entire system and can be executed in a limited time--usually a few hours--with limited data files. This approach is still quite expensive and is usually not used except in the case where very large computer systems are under consideration. The limited benchmark approach still requires a great deal of preparation involving both computer time and analysts' time.

Another expensive approach to performance planning is the use of simulation. This approach has several variations. In one variation, systems analysts build a model of the computer system from a basic computer language such as FORTRAN or PL/I. Another variation requires the analyst to build the model using a generalized simulation language such as GPSS, GASP, or SIMSCRIPT. The systems analyst sometimes chooses to use a basic system model supplied by a software vendor or consulting firm. In this case, the I/O configuration, memory size, and CPU speed of the target system are supplied as parameters to the basic system model. The major processes of each program are then modelled as events,i.e.

CPU access, I/O access, and "interruption" or pre-emption of the access to the CPU of a less important program (low priority) by a more important program (high priority). This method, while more practical than the first two approaches, requires purchase of the vendor's software or his consulting and educational services and can still be quite expensive.

In the real world, simulation has been used extensively to model large teleprocessing networks. Such networks are not typical of most data processing organizations. Even if the expense of the simulation approach were not prohibitive, many users of small and medium sized computers do not have employees with the skills necessary to do this kind of study, nor do they think that the simulation packages available are economically justified.

Scheduling

There is an urgent need need for easier or more convenient performance prediction. Another important area for which a predictive tool is required is computer scheduling. Computer scheduling, as used here, has to do with the sequences and combinations of programs which are determined externally to the computer, and is to be differentiated from schedules which are determined by the computer's operating system, which will be referred to as "task scheduling" or "dispatching". Scheduling, in the context used in this thesis, is normally a clerical function. The scheduler

tries to arrange the sequence of executions so that system resources are balanced and deadlines are met [23]. To do this, he must consider program memory size, peripheral requirements, job dependencies, data availability and other complex factors. The scheduler is really interested in predicting the actual job run time under a specific set of circumstances, but normally uses the average job run time that has been calculated over several executions of the job being scheduled.

Presently, most batch job computer scheduling is done using the average elapsed time for each job in the schedule [5,21], even in cases where the scheduling function is computer assisted. That is to say each program or sequence of logically related programs (job) is considered to execute for a fixed amount of time on a given computer system, regardless of the charactersitics of all the other programs in the system at the same time. The inadequacy of the deterministic type of scheduling can be observed from the non-deterministic nature of interactions between the jobs in a computer system at any given time and the effects of priorities.

Priorities and Interactions

Consider two programs which are assigned priorities or levels of importance such that the lower priority program will be forced to relinquish the CPU whenever the higher priority program is ready to access the CPU (usually at the completion of I/O activity). The lower priority program will have to wait for the higher priority program to relinquish the CPU and will only have potential access to the CPU during the times that the higher priority program waits for I/O operations. It is easy enough to visualize that if the low priority program always has to wait for the high priority program, and the high priority program never has to wait for the low priority program, the respective behaviors of the two programs will surely be affected if their roles are reversed.

Extending the example to the case of several programs will exaggerate the effects of program interactions in the processor. The actual run time for a particular execution of a given job or program will deviate from the calculated average elapsed time unless it is run with precisely the same set of other programs and identical assignment of priorities as when the average was calculated.

Interactions within the processor are further complicated by contention for channels and I/O devices. Returning to the two program example, suppose each program accesses a unique disk device and has no other I/O. Further suppose

that each program has a nontrivial disk utilization, 20% for example. If the data which these two programs access are consolidated on one disk drive, it is inconceiveable that the execution or run times of each of the two programs would be unaffected unless other (compensating) changes were also made.

There is another type of variation in job durations in a typical business oriented data processing system--the variation due to transaction volumes in transaction driven applications. For such applications the elapsed job times are proportional to the number of transactions processed if other things are equal. Since techniques such as trend analysis and exponential smoothing may be used for the prediction of transaction volumes, the focus of this research will be on the prediction of the more complex type of variation in job execution duration--the variation due to interactions among several programs executing in a computing system.

Batch versus Timesharing

Computer systems may be classified into two groups according to whether or not the system is primarily dedicated to batch multiprogramming or teleprocessing (time-sharing) multiprogramming. Within both groups, the dispatching scheme or the method which is used to allocate the resources of the computer system to competing programs or "tasks", may be a priority system, a time-sharing system or a combination

of the two. The priority dispatching scheme has the primary objective of optimizing the throughput (number of units of work completed per unit time) of the most important jobs or tasks. The time-sharing scheme, on the other hand, has the primary objective of optimizing (minimizing) the "response" time or the time it takes for each interaction of a terminal with the CPU. The combination scheme of priority dispatching within time-sharing is an attempt to have it both ways. Although either of the above dispatching strategies may be found in batch multiprogramming systems or in teleprocessing systems, the priority dispatching scheme tends to be associated with batch multiprogramming and time-shared dispatching tends to be associated with teleprocessing systems because this is consistent with the primary performance objectives of these respective systems. For this reason teleprocessing under the control of primarily batch multiprogramming systems may use priority dispatching and batch-oriented processing under the control of primarily time-sharing or "interactive" systems will be dispatched using the time-sharing discipline. Because most of the work on time-sharing systems is not under the control of the data processing installation but is initiated by the terminal user, the prediction of system loads is a statistical problem. It is useless to talk about scheduling a time-sharing system in the sense that "scheduling" is used in this thesis, therefore this research emphasizes primarily batch-oriented multiprogramming systems.

This investigation sets forth the development of an analytic model which is readily usable by computer system analysts, computer schedulers, and hardware and software vendors to predict gross computer system performance characteristics (CPU, channel, and device utilizations and throughput) as well as elapsed times and CPU, channel and device utilizations by program or job. A discussion of the background relevant to computer system performance prediction and related models is presented in chapter 1, and the model used in this research is developed in chapter 2. Chapter 3 consists of an explanation of the parameter estimation and validation prccedures and the experimental design. A presentation and discussion of the experimental results is given in chapter 4. The research results are discussed and recommendations for future research are presented in chapter 5.

I. BACKGROUND AND PREVIOUS WORK

CPU Models

Central Server Network Model

Most of the models developed to predict the performance of multiprogramming systems are closed, cyclic queuing network models (so-called "central server" models) based on the early work of J.R. Jackson [32] and later extensions by Gordon and Newell [28]. These researchers determined the conditions under which closed form solutions to network queuing models were known to exist. The types of networks which met these conditions were called "separable" and the solutions were said to be in "product-form". The product-form solution states that the equilibrium state probability for the network is the product of the equilibrium state probabilities of the component service centers in the network.

The central server model is based on the assumption that the execution of programs in a multiprogramming system consists of alternate periods during which each program is either receiving or waiting for CPU service and periods during which each program is either receiving or waiting for I/O device service. Another general assumption in the

central server model is that at the completion of CPU service, each program requests service from the i-th server (I/O device) with probability P(i). A schematic diagram of the central server model is given in Figure 1.

Simple Cyclic Queuing Model

One of the better known computer performance models is a model developed by D.P. Gaver, Jr. Gaver assumes an identical probability distribution for the CPU demand of each job and an identical exponentially distributed response time for each I/O device in the system [26]. Parameters in Gaver's model are the number of homogeneous jobs and their CPU service time and the number of homogeneous I/O units and their I/O service time. This model is a specific implementation of the central server model which has two stations, the CPU and the parallel server I/O station. The I/O devices in the Gaver model are treated as a pool from which a request for I/O may be serviced by any device which is idle. An arbitrary CPU service time may be modeled by either an Erlang, Hypoexponential or Hyperexponential distribution. Paging behavior is not explicitely modeled but may be considered to be included in the overall I/O rate.

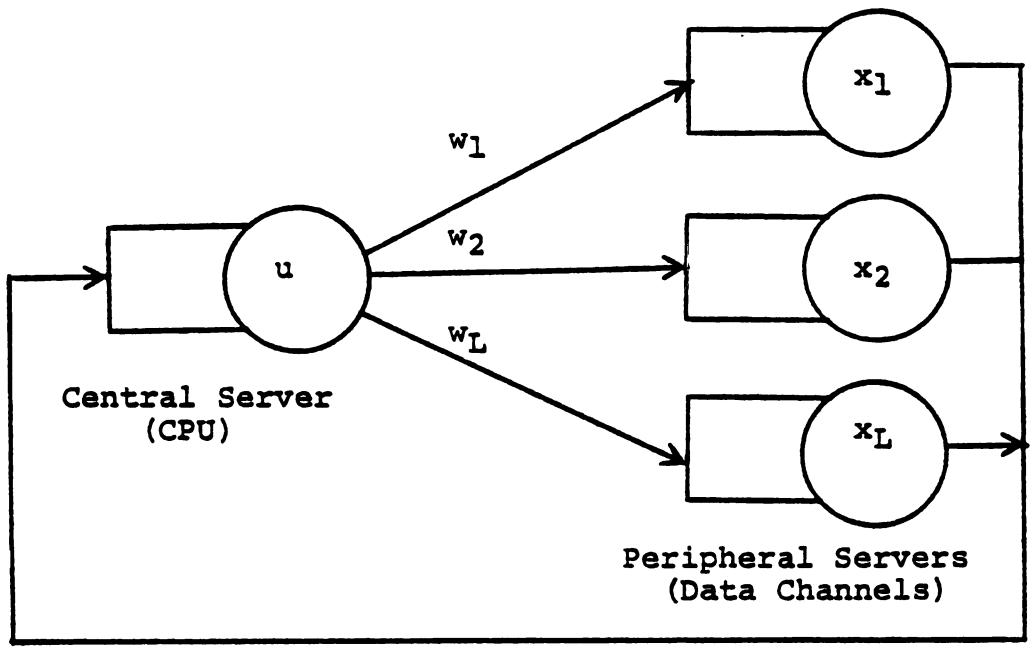

Figure 1. Central Server Model

Simple Flow Model

Fenichel and Grossman's Flow model [20] does not use probability distributions and does not account for direct program I/O but rather assumes a fixed relationship between average compute time and paging. For this model the only I/O considered is paging. The paging response is computed from a response table. Simulation of the operation cf the paging device is used to develop a table of response times under different paging rates. The Flow model makes nc assumptions about the form of the probability distribution of paging service time.

Cyclic Queuing with Paging and Overhead

Lewis and Schedler's Cylic Queuing model [34] returns to some of the central server assumptions and accounts for I/O and idealized paging behavior using exponential distributions. This model assumes that program execution intervals between page exceptions (requests for I/O) are identically and independently distributed exponential variables. In this way the dependence of paging rates on memory size is avoided. Like other central server models, this model considers the behavior of all programs in the system to be statistically identical. This is equivalent to partitioning the computer's main memory into equal sized segments and having the page replacement algorithm operate locally (each program would only steal pages from itself). If this were

not the case, independent execution intervals would not hold. This model differs from most of the central server models in that it explicitely includes the CPU overhead for task dispatching and paging I/O.

Product of Stages Model

Using some results by Cox on probability distributions with rational Laplace transforms [15], Basket and Gomez [4], and Muntz [15] extended the class of known closed queuing networks with product form solutions to include certain servers with general service time distributions by approximating the distribution with a combination of exponential stages (see Figure 2). Using the method of stages, closed form solutions are known to exist for: (1) exponential servers with first-come-first-served (FIFS) service discipline, (2) general servers with processor-shared (PS) discipline, (3) general servers with last-come-first-served-preemptive (LCFS) service discipline, and (4) infinite servers (IS) with general service distributions.

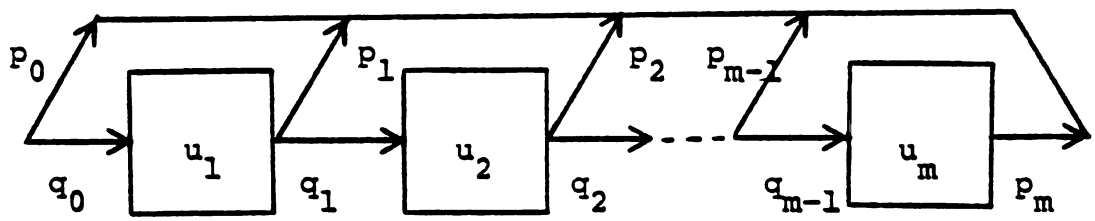

Figure 2. Rational Laplace Transforms

Multiple Resource Allocation Model

A Multiple Resource Allocation computer model has been developed by Boyd [7]. This model handles the resource requirements of the jobs in the multiprogramming mix in a similar fashion to most central server models. However, it goes much further in the sense that, given the level of multiprogramming, the probabalistic aspects of job selection for execution from the job queue are developed in great detail. The selection criteria is based on the permanent (execution) resource requirements of the jobs that are waiting to be added to the multiprogramming mix. Although this model is referred to as a batch multiprogramming model by the author, in reality it behaves very much like a time-sharing model. In fact, the dispatching strategy for this model is a time-sharing strategy. As a batch multiprogramming model, this model is representative of an installation where there are a very large number of small jobs, with few if any data preparation constraints, and no precedence constraints (by the assumption of independence). Futhermore there can be little if any external control of the job schedule since job selection and execution are entirely determined by statistical distributions.

"Straightfoward" Queuing Model

Another recent computer performance model has been developed by Boyse and Warn [8], and is a time-sharing cyclic queuing model. The CPU intervals in this model may be either constant or exponential and the only I/O modeled is paging. The CPU modeled has parallel paths to the I/O devices (drums) and, because the number of concurrently executing programs is small (3), the I/O response time may be treated as independent of the number of pending I/O requests. The assumptions in this model agree very closely with the features of the system for which it was developed, a dedicated graphics terminal system, and was found to have a very satisfactory fit.

Eclectic Model

A model by Brandwajn [9] has incorporated several recent developments into the central server model. Brandwajn includes paging in his model and uses Belady's "lifetime function" [6] to determine the effects of memory allocation on paging rates. Brandwajn also uses the "principle of decomposition" [14] to simplify the calculation of the equilibrium state probabilities of his model. The principle of decomposition states that if the elements of a subnetwork of the overall system have rates of state transitions much higher than the remainder of the system, this portion of the overall network will reach equilibrium sooner. This means that the subsystem composed of the CPU

and paging device may be separately analyzed under the assumption that the rate of paging is much higher than the rate of direct I/O. The total system is then modeled as a two-server system where one server is the disk I/O device and the other is the composite CPU-paging server.

Summary of Queuing Models

All of the above models are limited or inadequate for the purposes of this research because they all use global parameters to determine system behavior but say nothing about individual programs that may be executing at a given time. Futhermore they are really processor models that do not give a very realistic treatment to the input/output effects on the system. The assumption of homogeneity among I/O devices and channels is a serious weakness of these models for configuration and scheduling studies. The homogeneity assumption is tantamount to saying that different types of I/O devices aren't really very different therefore they can be treated identically. It is known that disk drives don't behave like tape drives, printers, or card I/O equipment. Since the latter devices are dedicated to a particular program at any point in time, the variation in their response times is usually due to the interference caused by devices used by other programs on the same channel. On the other hand, disks are normally shared among programs, thereby causing variation in response due to the potentially random positioning of the disk access arm before

I/O can take place as well as queuing time for both the device and channel. The variation in response introduced by shared disks is on the order of several times larger than the time required for data transfer. This research considers another approach which overcomes many of the objections to the central server models.

Synthetic Workload Benchmark

The Synthetic Workload method [42] of Sreenivasan and Kleinman gives good results but requires the solution of two or more simultaneous, non-linear equations in six unknowns for each job (program) being modeled. Implementation of the Synthetic Workload proceeds as follows. Let X1 represent CPU seconds and X2 the number of EXCP's (approximately equal to the number of I/O operations). Dividing each of these dimensions into L parts over the range of the X1 and X2 values for the actual workload, the percentage of observations in each cell of the total number of jobs will be the joint probability density of the real workload.

The synthetic workload may consist of a smaller collection of programs with the same joint probability function. If P(i,j) is the probability of the (i,j)-th cell and NTOT' the total number of programs in the synthetic workload, then the number of programs in the (i,j)-th cell of the synthetic workload is given by

$$N'(i,j) = P(i,j) * NTOT' \qquad i,j = 1,2,...,L. \qquad (1.1.1)$$

Then, if X1(i) corresponds to the mid-point of the i-th partition of the X1 dimension, the constraint on total CPU time for the synthetic workload is expressed as

$$sum[\ X1(i)*N'(i,j)\ :\ i,j = 1,2,..,L\ ] = T. \qquad (1.1.2)$$

The joint probability distribution of the real workload is duplicated by NTOT' executions of the same program, a synthetic program [10]. The synthetic program used by Sreenivasan and Kleinman simulates a file update process. Its execution characteristics may be manipulated by varying a set of six parameters supplied to the program by Job Control Language (JCL).

P1, P2, P3, P4, P5 and P6, the parameters of the synthetic program, correspond to the number of master records created, the number of detail records created, the number of executions of a "compute kernal" [33] per match of the master and detail files, the number of times the file update is repeated, I/O buffer blocksize, and record size respectively. The functional dependence of X1 and X2 on the six parameters can be expressed as

$$X1=K1+K2*P4+K3*(P1+P2)+K4*P4*(P1+P2)+K5*P2*P3*P4 \qquad (1.1.3)$$

$$X2=2*P4+(2*P4+1)*([P1*P6/P5]+[P2*P6/P5]). \qquad (1.1.4)$$

The constants K1, K2, K3, K4 and K5 may be estimated by regression experiments, but since there are more independent variables than equations, there is no unique solution to 1.1.2, 1.1.3 and 1.1.4. A solution is therefore achieved by choosing integral values for P1, P2, P5 and P6 and iterating on the values of P4 and P3 until the calculated and the "given" values for X1 and X2 agree. This must be done for each of the NTOT' programs in the synthetic workload.

A shortcoming of the synthetic workload is that it requires the actual setup and execution of the synthetic programs under the operating system being modeled. A further complication is the requirement that the target system be available for execution of the synthetic wcrkload. Although the version of the synthetic workload model discussed here can be extended to account for paging and I/O response time, such an extension will increase the computational complexity many-fold. Compared to the representative sampling approach, the synthetic workload does not require as much planning and preparation since only one program is involved and it generates its own data. For benchmarks in which the primary objective is comparison among alternative CPU's, the synthetic benchmark is superior since relative performance is the basis for decision. For benchmarks in

which system alternatives are being considered, the specifications, allocation and distribution cf data for the I/O subsystem becomes much more critical. In this case, extension of the synthetic model and more planning becomes necessary, thus negating some of the advantages of the synthetic workload model.

Both the representative sample benchmark and the synthetic benchmark share the disadvantage of being unsuitable for scheduling applications; the former because prediction involves actually running the job streams, and the latter because the synthetic jobs run approximately the same length of time as the real jobs.

Models of Paging Behavior

Working Set Model

Most models of paging behavior were developed as an aid in evaluating paging algorithms for operating systems. Denning's "working set" model [18,19] is defined in terms of the collection of pages of a program which have been referenced during the process interval [t-TAU,t], where t is an instant in time and TAU is an interval of time. Denning defines the working set size to be the number of distinct pages in the working set. He proposed the use of TAU by the operating system software (and hardware) as a parameter to control page residency.

Denning showed the working set size to be an increasing function of the parameter TAU and the page fault rate (the rate at which a program tries to access pages which are not present in main memory) to be equal to the negative slope of the working set size. The working set size function depends on some knowledge of the underlying probability distributions of the memory reference patterns.

The working set concept (as defined by Denning) has proven useful in the analysis of page management algorithms but does not serve well as a predictor of paging behavior for systems which do not use the working set parameter TAU to control page residency.

The Lifetime Function

A function proposed by Belady and Kuehner, the "lifetime function" [6], is based on a model of independent references to the pages in a program. The independent reference model assumes that each memory reference is an independent Bernoulli trial relative to each page of the program, where the probability of a reference to page i is given by q(i). Belady and Kuehner's independent reference model is a special case where q(i) = 1/S for a program with S pages. For this program, w pages in main memory results in a probability of (S-w)/S that a page reference will result in a page fault.

The lifetime function is defined to be the expected

number of consecutive references before a page will be referenced which is not in main memory (thereby causing a page fault). Using the geometric distribution, the lifetime function is the found to have the following form:

$$e(w) = w/(S-w). \qquad (1.2.1)$$

Belady and Kuehner then proceed to approximate the independent reference model by

$$e(w) = A*w**k, \quad 1.5<k<2.5 \qquad (1.2.2)$$

on the basis that real programs do not obey the independent reference model.

In this extension to the model, k is a function of the program's page reference behavior and A is a function of both the particular processor in which the program is executing and the program reference patterns. The factor A may be further decomposed into the product of K, the average instruction execution rate cf the processor, and A', the average page reference per instruction.

Markov Models

Oden and Shedler developed a model of paging behavior which is based on a Semi-Markov process [38]. They define equivalence classes for the states of their model to reduce the state space and present a solution for the steady state probability distribution of the reduced state space. They assume that the transition probabilities for each of the reduced states is known and that the page frame (page of real memory) to be "stolen" for a paging operation is selected randomly. They alsc assume that the N programs in the multiprogramming set are dispatched First-In-First-Out (FIFO) and that they are statistically identical.

These last few assumptions make this model unsuitable for use in modeling a priority dispatched operating system with a specific workload. The assumptions make this particular model more useful for a time-sharing system, and for investigating paging behavicr with regard to the determination of the optimal value for N, the multiprogramming level.

The page fault rate, steady state page residency and page fault probability have been modeled as a Markov process by Franklin and Gupta [24]. Using page transition diagrams, FIFO and Least-Recently-Used (LRU) paging algorithms, they developed a memory state transition matrix which could be used to determine paging statistics.

While the foregoing approach is an interesting tool in

the examination of paging behavior and page replacement and selection algorithms, it is defined in terms of a single program's behavior and requires exact knowledge of the transition probabilities for each page of a program. Direct measurement of the variables necessary to compute the transition probabilities of each program in an operating system would impose too high a cost in system overhead. Of course, it is possible for hardware to be designed to achieve this function, but the economics of such hardware is extremely doubtful. What can be easily measured by hardware or software is occupancy, the amount of time spent executing the instructions of a given page. Such an approach might be used if one is willing to assume that the page occupancy probability is a sufficient proxy for the actual transition probability (this is equivalent to making the page transition probabilities for each page identical). Knowledge of the occupancy probabilities does not yield enough equations to estimate transition probabilities for each page of a program. A final objection to this model is that the computational complexity and system overhead associated with it makes it impractical for all but very small, i.e. pedagogic examples.

A "Half-Life" Model

A paging model that is based on fitting a lifetime function to the shape hypothesized by Belady and Kuehner was proposed by Chamberlain, Fuller and Liu [12]. This function has the following form:

$$e(w) = 2*B/(1+(C/w)**2), \qquad (1.2.3)$$

where w is the number of pages a program has in real memory and B and C are its paging parameters. B is defined as one-half the largest possible lifetime and C is the number of resident pages which provides the process with a lifetime of B.

The Simple Linear Model

A simple linear model of demand paging performance proposed by Salter [40] is based on the assumption that the mean number of consecutive page references before a page fault (exception) occurs is linearly propcrtional to the size of main memory (Saltzer refers to this as "headway" rather than lifetime). Saltzer's graphs for associative memory headway show a significant deviation from linear at small memory sizes and his graph of paging in the MULTICS system has measurements at only two memory sizes, hardly enough to determine curvature. He quotes three sources of published measurements which report the lifetime (or headway) increasing faster than linear.

Saltzer's model is stated in terms of the entire main memory of the computing system and he asserts that the model does not represent the behavior of a single program and does not even predict headway at extreme values where paging approaches zero. He then goes on to present examples of individual program paging prediction by assuming an identical distribution of main memory pages to all processes in the system.

The Paging Index

Another paging model which predicts system-wide paging statictics is Bard's Paging Index model [3]. The Paging Index (PI) is an emperically derived model that uses estimates of working set sizes to predict system paging rates. The working set size w is estimated externally to the model. An estimate of working set size w, is made by software monitors for each logged-on user of IBM's VM/370 time-sharing system. Given a pageable main memory size of M, and the average number of logged-on users N, the storage saturation factor S is given by

$$S = N*w/M. \qquad (1.2.4)$$

The paging rate is then estimated by

$$P = I*(S**2)/4 \qquad 0 \leq S \leq 2$$
$$= I*(S-1), \qquad 2 \leq S \qquad (1.2.5)$$

where the single parameter I characterizes the entire P versus S curve.

Bard's measurements of several systems supports the apparent deviation from the linear model in some of Salter's diagrams of paging measurements. Like Salter's model, this model is not very useful for predicting the behavior of a specific program although it has proven useful in predicting average behavior for a composite or "typical" program in a time-sharing environment under the assumption that the probability distributions of the transactions (programs) arriving from the terminals in the system are statistically identical.

The Page Survival Index

Bard's Page Survival Index [1,2] is typical of a class of paging models that is based on dynamically computed statistics. The Page Survival Index (PSI) is a measure of an unreferenced page's ability to survive interruptions in the operating system without being selected for replacement. Since a program is most likely to lose pages while it is in an interrupted state, Bard considers the PSI to be a very good representation of an individual program's paging behavior.

The definition of PSI requires the measurement of several dynamic paging statistics for its estimation. The PSI is used by Bard as a response variable in the control of the page management system and scheduler by feedback. This model does not lend itself to prediction since it requires the continuous calculation of dynamic variables.

Of all the paging models presented, most do not serve well as models of paging behavior for performance prediction since they were developed for use in controlling the paging process with feedback [19,1,2].

I/O Models

As most models of I/O behavior were developed as a part of multiprograming models, there are few general models of I/O behavior. Of the I/O models that do exist, most are simple exponential models of non-specific I/O devices. The closest thing to I/O models are models for the investigation of disk scheduling policies and the effects of various disk organizations.

Disk Response Model

Seaman, Lind and Wilson analyzed disk I/O as an integral part of overall teleprocessing system design [41]. They assumed Poisson arrivals for all disk requests and they also assumed equal traffic to each disk module in the configuration. For the disk service times they assume

identical but arbitrary probability distributions. They do make some suggestions as to modifications to their model in order to account for unequal traffic to the disk modules.

Disk and Drum Scheduling Models

Denning developed models of both disk and drum file systems to study the effects of different scheduling policies on the response times of direct access devices [18]. Among the scheduling policies Denning investigated were: (1) Shortest- Seek-Time-First (SSTF), (2) Shortest-Access-Time-First (SATF) for drums, (3) First-Come-First-Served (FIFO), and (4) SCAN, which involves sweeping the disk access arm back and forth across the surface of the disk, stopping at any cylinder for which there are requests. He concluded that SATF was the optimal policy for drum scheduling, and that the SCAN policy was superior to FCFS, which is in turn superior to SSTF. Many of these ideas have been incorporated in today's operating systems.

Teory studied the same scheduling policies as Denning, but he added variations to the SCAN policy [43]. The Circular Scan (C-SCAN) policy involves serving disk requests only while the acccess mechanism moves in one direction (usually from the outer cylinders toward the inner cylinders). The N-step Scan (N-SCAN) allows requests to be serviced while the disk arm moves in both directions but all requests which arrive while the arm is sweeping in any direction is batched for service during the return sweep.

Assuming uniform I/O request distributions, Teory found the C-SCAN policy to be superior at I/O rates greater than 40 requests per second and the SCAN policy to be superior at rates less than 40 requests per second. The N-SCAN was found to be worse than SCAN or C-SCAN at all rates of I/O.

The Disk Seek Model

Waters [44] derived formulas for average seek distance and average seek time for both sequentially and randomly accessed disk files (under the assumption of uniform distribution of accesses). He also derived formulas for computing the average seek distance and time for files that do not have uniformly distributed random accesses. Waters demonstrated that the average seek distance between two files is the difference between their mid-points and that the file access time is minimized by placing the highest activity records of a randomly accessed file at the center of the file.

Revised Disk Response Model

Wilhelm elaborated the model developed by Seamon, et al. in a general disk performance model in which he assumes neither a uniform distribution of workload over all disk modules (spindles) nor a uniform distribution of accesses over any disk module. Like most other models of I/O behavior, this one assumes that the requests for disk I/O are generated by a Poisson process. Wilhelm places no

restrictions on the service time distributions other than the requirement that their Laplace transforms exist.

Of the models of disk I/O mentioned above, the Seaman, et al. and Wilhelm models are perhaps the most useful for the purposes of this research. Denning's model and Teorey's model are of use in understanding disk scheduling but are less helpful since the operating system used in this research has not implemented SCAN disk scheduling. Although Waters' article was written for the file designer and has a practical orientation, some of his techniques were applied in the elaboration of the I/C submodel presented in this thesis.

II. MODEL DEVELOPMENT AND ANALYSIS

Batch Model Requirements

For the purposes of this research, an analytical model is required which, in some way, accounts for priorities, system overhead and paging as well as a normal configuration of I/O devices. The analytical model is required because of the necessity of having a performance predictor which can be used for external scheduling. Simulation or synthetic benchmark techniques are too time consuming for this purpose although they are very well suited to performance analysis in connection with a major equipment acquisition. Another reason why there is a real need for the type of model described in the remainder of this section, is that the flexibility of data processing management should not be limited by the necessity of authorizing a major study in order to be able to answer relatively simple "what if" questions from top management.

The model should be usable by typical data processing systems personnel without special training and should use system captured data to generate the program related characteristics used in prediction. In addition, computer operations management and systems programmers can benefit from a

straightforward tool for studying the effects of scheduling changes and operating system parameter changes in their efforts to run a "near optimum" operation. A necessary requirement for scheduling is that the model predict the performance of individual programs operating in a multiprogramming mix. Furthermore, the predictor should operate with a set of program characteristics which are very nearly invariant under different operating systems or hardware configurations.

Such characteristics are known to exist within a family of computers such as the IBM System/370 line using either the Disk Operating System for Virtual Storage (DOS/VS), or the Operating System for Virtual Storage I (OS/VS1). This is true because these operating systems are enough alike that differences in execution characteristics of a program compiled under these two systems would be mainly reflected in CPU overhead, since the structure and dispatching scheme of these two systems are very nearly the same. In other words, the differences between these two systems can be reflected by differences in the parameter values of the basic model.

Since the purpose of this model is the prediction of the performance of an existing set of programs on a computer system which may be different from the system on which the programs are currently executing, or the prediction of the performance of each program in a collection of programs

under a specific set of conditions (scheduling), the required characteristics can be estimated from the data that job accounting routines normally collect in most operating systems. The data collected for each execution of a program includes: (1) the number of I/O operations per execution, (2) the number of I/O operations by device, (3) average block sizes of data transferred by device or file identification, (4) path length or the approximate number of instructions processed per program execution (see M. Reiser:39), (5) job elapsed time, (6) system wait time, (7) number of page-in and page-out operations, and average working set size. The data captured by the job accounting routines may then be used to estimate the mean CPU service time, the mean I/O service time by device, and the probability of I/O to each file or device following a CPU service. These statistics may be estimated for each program which is executed on the base system.

Program statistics as defined above can be used with system parameters such as configuration, hardware speeds, memory size and instruction execution rates to determine the performance and duration of each job in a given mix. Program parameters which are <u>not</u> invariant with respect to the configuration, hardware speed or multiprogramming mix may be expressed as functions of these factors. For example, the paging rates for programs operating in virtual systems depends upon the speed of the CPU, speed of the paging device(s), and the memory reference patterns of all

the programs in the system [18,27].

The model consists of three submodels: (1) a CPU submodel, (2) a Paging submodel, and (3) an I/O submodel. The CPU and I/O submodels are cylic queuing models [25,35,46] and the paging submodel is a theoretical model based on independent references and observed program paging behavior.

CPU Submodel

The CPU submodel used in this research is based on a priority interrupt driven dispatching scheme. One problem with modeling this type of system is caused by the fact that priorities are accounted for. Another difficulty is that a finite number of sources in a queuing system is more difficult to model than an infinite source system. The classical "machine repairman" model fails because the behavior of each program in the system cannot be considered to be statistically identical. This is true because of the requirement of predicting individual program behavior for a priority dispatching operating system.

Because of the intractibility of finite source models, infinite source models were used to approximate the finite source models. A program's execution interval or elapsed time is divided into alternating periods of CPU and I/O activity. Situations which violate this condition (such as

double buffering or overlapping CPU and I/O activity for the same program) were excluded from the analysis. The CPU phase may be thought of as being broken up into two distinct subintervals; the mean "initial wait" interval during which the program is waiting for the CPU following I/O, and the mean "completion" interval which is total elapsed time from the instant at which the program gains access to the CPU until it relinquishes the CPU for an I/O operation or program termination [25]. These two intervals are designated W and C respectively. The mean interval during which the program is waiting for I/O completion is designated as IO, the inverse of the instantaneous access rate LAM, which is assumed to be fixed in the CPU submodel. The mean CPU "cycle", ET is the sum of W, C and IO (see Figure 3).

CPU Completion Time

The completion time C, consists of the CPU quantum attributable to the program being considered B, plus the system overhead involved in task switching, initiating and terminating I/O for this program. The completion time also includes all the time which the program in question spends waiting for the CPU after it has been pre-empted by a higher priority program.

We designate the priority level as well as the identity of each program by the index j, where lower values of j represent higher priorities. The values for j are, of course, limited to positive integers.

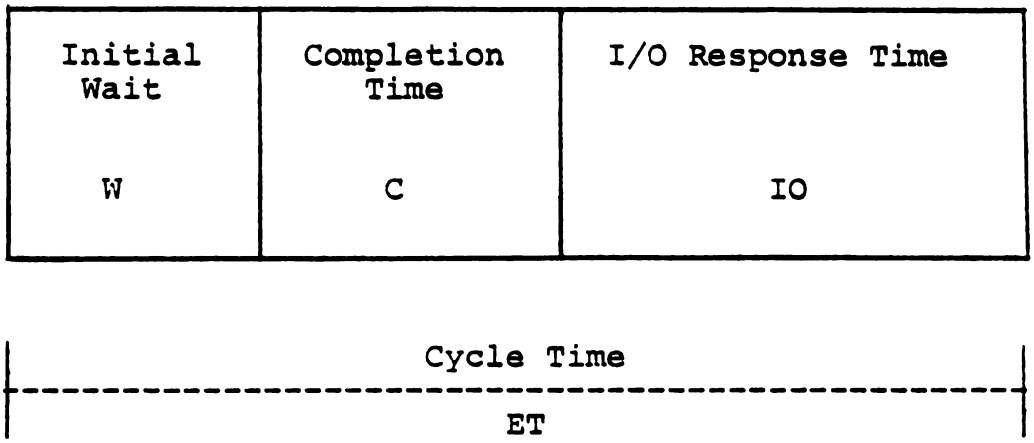

Figure 3. Elapsed Time "Cycle" (ET)

The average CPU time quantum may be estimated from the quotient of program state time divided by 1 plus the number of I/O operations issued by a given program. This quantity is represented by 1/MU(j) fcr program j. MU is the instantaneous CPU access rate in the absence of paging, and B(j) is defined as the mean execution interval before any I/O (including paging) therefore B(j) is less than or equal to 1/MU(j).

The total CPU quantum attributable to task j for each cycle is represented by TAU(j), where TAU is composed of B(j) plus an expression for the CPU overhead due to task switching, initiating and terminating I/O operations. It is assumed that TAU(j) is exponentially distributed.

In deriving the CPU submodel program 1 will be considered first. The completion time fcr program 1 is unaffected by any other task since this task has the highest priority. The exception where processing by a higher priority task is interrupted to handle the completion of I/O by a lower priority task is ignored. The completion time for task 1 is given by

$$C(1)=TAU(1). \qquad (2.2.1)$$

Next, program 2 is considered. Its service time will be TAU(2), but this may be spread over a longer interval if program 2 is pre-empted by program 1. While program 2 is

using the CPU, program 1 must be waiting for I/O completion. From the definition of the I/O completion time, I/O completions occur at the mean rate LAM(1).

On the average LAM(1)*TAU(2) I/O operations for program 1 occur during one "completion" time for program 2, assuming that the I/O completion time is exponential. Service time for program 1 will then cause a delay of LAM(1)*TAU(2)*C(1) in the service of program 2. The completion time for program 2 will be given by

$$\begin{aligned} C(2) &= TAU(2)+LAM(1)*TAU(2)*C(1) \\ &= TAU(2)*(1+LAM(1)*C(1)) \\ &= TAU(2)*(LAM(1)+1/TAU(1))*C(1). \end{aligned} \qquad (2.2.2)$$

At this point a new element is introduced. It is now possible for program 2's I/O to complete while program 3 is interrupted by program 1. The maximum number of I/O completions by program 2 during this interval is the minimum of LAM(2)*LAM(1)*TAU(3)*C(1) and LAM(1)*TAU(3) since there can only be one I/O completion by program 2 during a completion time for program 1. This quantity is represented by H(2)*TAU(3). We then have for program 3

$$C(3)=TAU(3)+LAM(2)*TAU(3)*C(2)$$
$$+LAM(1)*TAU(3)*C(1)+TAU(3)*H(3)*C(2) \qquad (2.2.3)$$
$$=TAU(3)*(LAM(2)+1/TAU(2)+H(2))*C(2).$$

Proceeding by induction and using similar logic, it can be shown that

$$C(j)=TAU(j)*(LAM(j-1)+1/TAU(j-1)+H(j-1))*C(j-1)$$
$$H(j)=\min[LAM(j)*(C(j)/TAU(j)-1),D(j)]$$
$$D(j)=D(j-1)+LAM(j-1)+H(j-1) \qquad (2.2.4)$$
$$\text{for } H(1)=D(1)=0 \text{ and } j=2,3,...$$

Using the definition of completion time C, and assuming exponential distributions for CPU service time TAU and I/O response time IO, we approximate the expected value for the completion time squared to compute the coefficient of variation of completion time and thereby the follcwing approximation for FAC which will be needed to compute the waiting time W.

$$FAC(j)=FAC(j-1)*[1+(2*(LAM(j-1)/TAU(j-1)$$
$$+H(j-1)*(LAM(j-1)-1/TAU(j-1)))) \qquad (2.2.5)$$
$$+H(j-1)**2)*(C(j-1)*TAU(j)/C(j))**2]$$
$$\text{with } FAC(1)=1.$$

CPU Waiting Time

Derivation of the initial wait, W is not so easy since we do not have a concept similar to completion time tc work with in this finite source queuing process. We begin by considering the two states, CPU busy and CPU idle with respect to program j. Again we need not be concerned about programs with index greater than j since they can be pre-empted by program j. With j=1 we immediately have W(1)=0 since there are no higher priority programs.

Next, we consider W(2). Here we assume that CPP(1), the CPU utilization due to program 1, has been previously computed. Since the only busy time that can affect program 2 is caused by program 1, we have

$$W(2)=CPP(1)*FAC(1)*C(1). \qquad (2.2.6)$$

FAC(1) is a scale factor which represents the portion of C(1) which is the expected wait time for program 2 when it finds the CPU being used by program 1. FAC is related to the coefficient of variation (Cs**2) for completion time by the expression FAC(j)=(1+Cs(j)**2)/2.

Deriving the expression for the wait time for program 3, we must account for the expected wait time due to program 1, program 2, and the wait time incurred due to the ccmple-tion of program 2's I/O while waiting for program 1 tc relinquish the CPU. We designate the latter quantity by

T(2)*C(2)/LAM(2)*ET(2), where T(2) = CPP(1) * min[LAM(2)*C(1),1]. The rationale for this expression is that if program 3 completes during an execution interval for program 1 and program 2 is in the I/O stage, then a maximum of one I/O completion by program 2 can take place and LAM(2)*C(1) will take place if this number is less than one. Assuming that the cycle time for program 2 (ET(2)) has been computed, the probability that program 2 is in the I/O stage is approximately 1/LAM(2)*ET(2). Thus we have

$$\begin{aligned} W(3) &= CPP(2)*FAC(2)*C(2)+CPP(1)*FAC(1)*C(1) \\ &\quad +T(2)*C(2)/LAM(2)*ET(2) \\ &= W(2)+(CPP(2)*FAC(2)+T(2)/LAM(2)*ET(2))*C(2) \end{aligned} \qquad (2.2.7)$$

For W(4) we have

$$\begin{aligned} W(4) &= CPP(3)*FAC(3)*C(3)+CPP(2)*FAC(2)*C(2) \\ &\quad +CPP(1)*FAC(1)*C(1) \\ &\quad +T(2)*C(2)/LAM(2)*ET(2) \\ &\quad +T(3)*C(3)/LAM(3)*ET(3) \end{aligned} \qquad (2.2.8)$$

where T(3)=CPP(1) * min[LAM(3)*C(1),1] + CPP(2) * min[LAM(3)*C(2),1]. By applying 2.2.6 and 2.2.7 and gathering terms we have

$$W(4) = CPP(3) * FAC(3) * C(3) + W(3) + T(3) * C(3) / LAM(3) * ET(3) \qquad (2.2.9)$$
$$= W(3) + (CPP(3) * FAC(3) + T(3) / LAM(3) * ET(3)) * C(3)).$$

Applying an inductive argument, it can be shown that the following expressions may be used to compute the mean initial wait interval:

$$W(j) = W(j-1) + (CPP(j-1) * FAC(j-1) + T(j-1) / LAM(j-1) * ET(j-1)) * C(j-1) \qquad (2.2.10)$$
$$T(j) = sum[CPP(l) * min[LAM(j) * C(l), 1] : l = 1, .. j-1]$$
$$W(1) = T(1) = 0 \quad \text{for } j = 2, 3, \ldots$$

CPU Utilization and Elapsed Time

With W(j) and C(j) determined and IC(j) considered fixed in the CPU submodel we have but to determine the mean elapsed time per cycle ET(j), and the CPU utilization CPP(j). We have the following definitions:

$$ET(j) = C(j) + W(j) + IO(j), \qquad (2.2.11)$$

$$CPP(j) = TAU(j) / ET(j), \qquad (2.2.12)$$

$$CP(j) = B(j) / ET(j), \qquad (2.2.13)$$

and

$$COH(j) = CPP(j) - CP(j). \qquad (2.2.14)$$

CP(j) and COH(j) represent program j's mean "problem state" utilization and "system state" (overhead) utilization respectively.

Paging Submodel

The paging model was developed by fitting a curve to observed paging behavior of programs in virtual systems, but turns out to be identical in structure to Belady and Kuehner's independent reference model. It may be observed that the paging rate of a program in a virtual system is inversely proportional to the amount of memory allocated and directly proportional to the difference between the size of the program and the amount of memory allocated (provided the difference is not negative). Furthermore, the instantaneous paging rate increases unboundedly as the amount of real memory allocated to the program approaches zero, and decreases to zero as the amount of memory allocated approaches the actual size of the program. Representating the amount of memory required by the program to execute without paging by S, and the mean amount of memory allocated by w, the paging rate is then proportional to the maximum of zero and (S-w)/w.

Page Exception Rate

Aside from the effect of a program's own working set size on its paging rate, the working sets of all programs in the system taken together have an effect on each program. This effect is similar to Bard's Paging Index [3]. The "critical memory" M(j) is defined as the size of program j's working set below which the program will begin to do non-trivial paging (above .5 pages per second) provided CPU cycles are available. Defining constants of proportionality K as a function of CPU instruction execution speed, A as a function of the number of memory references per instruction, and R as the total pageable memory, the instantaneous page exception rate is estimated by

$$CER(j) = \max[0, K*A(j)*(S(j)-w(j))*EXP(N)/w(j)], \qquad (2.3.1)$$

$$EXP(N) = [\text{sum}[M(l) : l = 1,,N]/R]^{**}q.$$

The real-time page exception rate PI is estimated as

$$PI(j) = CER(j)*CP(j), \qquad (2.3.2)$$

where CP(j) is the portion of CPU utilization attributable to program j not including system overhead. The assumption is that system overhead processing is performed using fixed, non-pageable memory.

Memory Allocaton

The problem now is to allocate the pageable memory R, to the N programs considered to be in the multiprogramming mix. Of the many possible ways to achieve this partitioning, the most reasonable appears to be to partition the memory in a way that minimizes the total system paging rate. The problem may be stated as follows:

$$\text{minimize} \quad FN=\text{sum}[\,PI(j) : j=1,2,..N\,] \tag{2.3.3}$$

$$\text{subject to} \quad \text{sum}[\,w(j) : j=1,2,..N\,] < R \tag{2.3.4}$$

$$\text{and} \quad w(j) \leq S(j) \text{ for } j=1,2,..N. \tag{2.3.5}$$

Except for not being differentiable at the constraints w(j)=S(j), this problem satisfies the Kuhn-Tucker conditions. Applying the knowledge which was developed about the form of the expression for PI(j), a solution may be derived.

Ignoring the individual program size constraints (2.3.5) for the moment, and applying the Kuhn-Tucker theorem, we have the following minimization conditions:

$$DFN/dw(j) = -K*A(j)*CP(j)*S(j)*EXP(N)/w(j)**2, \tag{2.3.6}$$

$$DFN/dw(j) = DFN/dw(i) \quad \text{for } i,j=1,2,..N, \tag{2.3.7}$$

where DFN/dw is understood to represent partial

differentiation. These conditions reduce to the following solutions

$$SUM=sum[(A(i)*CP(i)*S(i))^{**}.5:\ i=1,2,...N], \qquad (2.3.8)$$

$$w(j)=R*(A(j)*CP(j)*S(j))^{**}.5/SUM \quad \text{for } j=1,2,..N, \qquad (2.3.9)$$

$$w(i)=R*w(j)*(A(i)*CP(i)*S(i)/A(j)*CP(j)*S(j))^{**}.5 \quad \text{for } i,j=1,2,..N. \qquad (2.3.10)$$

If all the constraints $w(j)<S(j)$ are satisfied, we have the optimal solution since any deviation from this solution will cause an increase in FN. If we suppose otherwise, a rearrangement of the memory balance between any two programs would result in $DFN/dw(j)<DFN/dw(i)$ and the only change that would produce a decrease in FN would be to subtract memory from program i and add it to program j. However, this would only yield the initial solution.

If the memory constraints are active for some of the programs, i.e. $w(i)=S(i)$, no reallocation that adds memory to such a program could cause a decrease in FN since they are already at the minimum page rate (zero). However removing memory from a program that does not have the memory constraint active will result in an increase in the paging rate.

On the other hand, if the initial solution to the

system results in the situation w(j)>S(j), for all programs j, we are finished and we simply set w(j)=S(j) since the composite constraint is not binding, i.e. sum(w(j) : j=1,2,..Ni is less than available pageable memory R.

The fourth case is the most interesting (and probably the most common in practice). In this case at least one of the programs' memory constraint is violated and at least one is not binding. Let Q=[j : 1≤j≤N, w(j)≥S(j)] and let H=[j : 1≤j≤N, w(j)<S(j)]. For every j in Q we set w'(j)=S(j), compute R1=sum[w(j)-S(j) : j in Q] and reallocate R1+sum[w(j) : j in H] to all the programs whose indexes belong to H.

Let us re-examine the set [w(j) : j in H]. We see that minimization of the paging rates over the set [w'(j) : j in H] subject to sum[w'(j) : j in H]=sum[w(j) : j in H] yields the same set of solutions as minimization over <u>all</u> <u>N</u> of the program working sets.

We have the following solution

$$w'(j)=\text{sum}[\,w(i):i \text{ in } H]*(A(j)*CP(j)*S(j))**.5/SMH \quad \text{for } j \text{ in } H, \tag{2.3.11}$$

where SMH=sum[(A(i)*CP(i)*C(i))**.5 : i in H]. Applying the equilibrium condition

$$w'(j)*(A(i)*CP(i)*S(i))**.5=w(i)*(A(j)*CP(j)*S(j))**.5$$

we have

$$w'(j)=sum[\,w(j)*(A(i)*CP(i)*S(i))**.5:\ i \text{ in } H]/SMH$$
$$=w(j) \qquad j \text{ in } H. \qquad (2.3.12)$$

It is clear that, if we add the amount R1 to sum[w(j) : j in H], all of the w'(j) for j in H will increase and thereby decrease the value cf the partial derivative DFN/dw. Thus we have w'(j)>w(j) for all i in H, implying that

$$DFN(w'(j))/dw(j)<DFN(w(j))/dw(j)=DFN(w(i))/dw(i) \qquad (2.3.13)$$

$$DFN(w'(j))/dw(j)<DFN(w(i))/dw(i)<DFN(S(i))/dw(i) \qquad (2.3.14)$$
for j in H and i in Q.

This reassignment cannot produce a situation where any reallocation from the set [w'(i) : i in Q] to the set [w'(j): j in H] can result in a reduction in FN. We can drop the former set from further consideration and proceed as we did initially. Applying the algorithm to the set [w'(j): j in H], we must have either all constraints binding, no constraints binding, or a combination in which case the foregoing logic is again applied. We eventually arrive at a stage where either Q or H is empty and the algorithm is terminated.

"Near-Optimal" Memory Allocation

Because the paging mechanisms used by most existing operating systems are not optimal we may modify this procedure to reflect more realistic operating system behavior. For example, the paging algorithm may be biased toward the higher priority programs in the type of operating system for which this analysis is intended. We may wish to define a scale factor, L(j)=(1/m)**j*a, where m is greater than or equal to 1 and a is greater than or equal to zero. The Kuhn-Tucker conditions then lead to the following:

$$A(j)*CP(j)*S(j)*L(j)*w(i)**2=A(i)*CP(i)*S(i)*L(i)*w(j)**2 \quad \text{for } i,j=1,2,..N. \tag{2.3.15}$$

An example of a biased paging model is one that yields the constant ratio w(j+1)=p*w(j) provided that p is greater than zero and less than or equal to 1, and that A(j+1)*CP(j+1)*S(j+1)=A(j)*CP(j)*S(j). This leads to the parameter L(j)=p**(j-1) and the solution is

$$w(1)=(1-p)*R/(1-p**N), \tag{2.3.16}$$

$$w(j)=w(1)*p**(j-1). \tag{2.3.17}$$

In parameterizing the paging submodel L(j) is one of the values that we could estimate. The actual I/O estimates for the paging model will be calculated by the I/O submodel which will now be developed.

I/O Submodel

I/O Overview

The I/O model used in this research is the composition of a number of simpler models of assumed independent components. The I/O response time of the composite model will be computed as a sum of random variables with uniform, exponential and Erlang distributions. This model will be elaborated primarily as a disk I/O model since disk activity is the predominant form of I/O activity in most modern operating systems and disk I/O is also that aspect of the system which causes the most difficulty in modeling. Disk I/O is so prevalent because disks are used for paging devices, data staging devices (temporary storage), and permanent storage devices. The Unit Record (card and print) activity is generally "spooled" on disk. This means that card input and output as well as print operations are actually disk operations during user program execution, with the card input to disk taking place prior to program initiation and card and print output taking place after program termination. The Unit Record (U/R) devices are driven by special system programs called "spoolers" or "readers" and "writers".

I/O Submodel Development

The operation of the I/O submodel is related to the CPU and Paging submodels as follows: The I/O wait time in the CPU submodel is actually the average wait time over all the forms of I/O in which the program engages, including paging. The I/O access rates generated by the CPU submodel are composites of the rates for each program's files. The device access rates are the aggregate of the access rates by program. A program's access rate to a given device is the product of the program's total number of accesses to that device times the program's composite access rate divided by the program's total number of I/O accesses.

Let the number of accesses to file k on device i by program j be given by N(i,j,k). Then the total number of I/O operations by program j is

$$N(j) = \text{sum } N(i,j,k):\ i=1,2,..ND;k=1,2,..NF] \quad j=1,2,..N, \qquad (2.4.1)$$

where ND is the total number of devices and NF is the number of files on device i. Since the number of accesses by program j to device i is N(i,j), the access rate to device i by program j is then

$$ARAT(i,j) = N(i,j)/N(j) * ET(j)$$

$$i=1,2..ND \text{ and } j=1,2,..N. \qquad (2.4.2)$$

The total access rate to device i is then

$$ARAT(i) = sum[ARAT(i,j): j=1,2,..N]. \qquad (2.4.3)$$

To simplify the computations, the arrival rate of I/O requests at the I/O device queues is assumed to have a Poisson distribution. Even if it was assumed that both the CPU and the I/O devices were exponential servers, there would not be Poisson arrivals in general because the CPU uses a priority-resume service discipline [32].

For the case of card or print I/O we will assume a constant service time. The U/R service time will depend on the number of cards or lines transferred (to disk) per I/O operation and the speed of the I/O devices. The only variation in the response time for these devices will be that due to channel contention. A program which issues an I/O operation to a U/R device will only have to wait on the channel. Because these devices are dedicated, a program accessing them will never find them busy since we have excluded the possibility of buffered operations.

Another type of dedicated device is magnetic tape. In this case the mean service time depends on the mean block-size of the data transferred and on the tape transfer speeds of the magnetic tape devices. In addition to channel delay,

we will also experience control unit delay. The latter form of delay is included with channel delay or ignored as negligible since the configurations to which this research applies rarely have more than one tape control unit per channel.

The channels to which U/R devices, tapes and disks are attached are almost always unique so that these components of the I/O subsystem may be treated separately in computing the I/O response.

The most complex portion of the I/O subsystem is the part which deals with magnetic disks. It is assumed that the size, location and disk identification for each logical file accessed by the programs in the system are known (either from job accounting data or otherwise).

Disk I/O Model

Three basic patterns of disk access are considered: (1) uniformly distributed random access over the cylinders of a file, (2) uniformly distributed indexed access over the cylinders of a file (requiring prior access to an index before accessing the data record), and (3) sequential access. A disk access mode diagram appears in Figure 4. This is only a partial diagram of the variations on the three major modes of disk access. In what follows, fixed record (block) sizes, "verify" option for all writes, a uniform distribution for rotational delay, and a uniform

distribution of accesses for random access to both random and indexed files is assumed.

It should be noted that some of the equations developed in this section are hardware or implementation dependent. In particular the access methods are IBM implementations and the disks are IBM 3330 disks. These hardware dependencies will be pointed out where applicable.

A timing diagram for indexed access appears in Figure 5, and one for random access in Figure 6. The diagram for sequential access difffers from the latter only by the absence of a significant seek time component. In what follows, the occurance of equations with indices i,j and k can be assumed to imply that the equations will hold for the entire range of each index unless stated ctherwise.

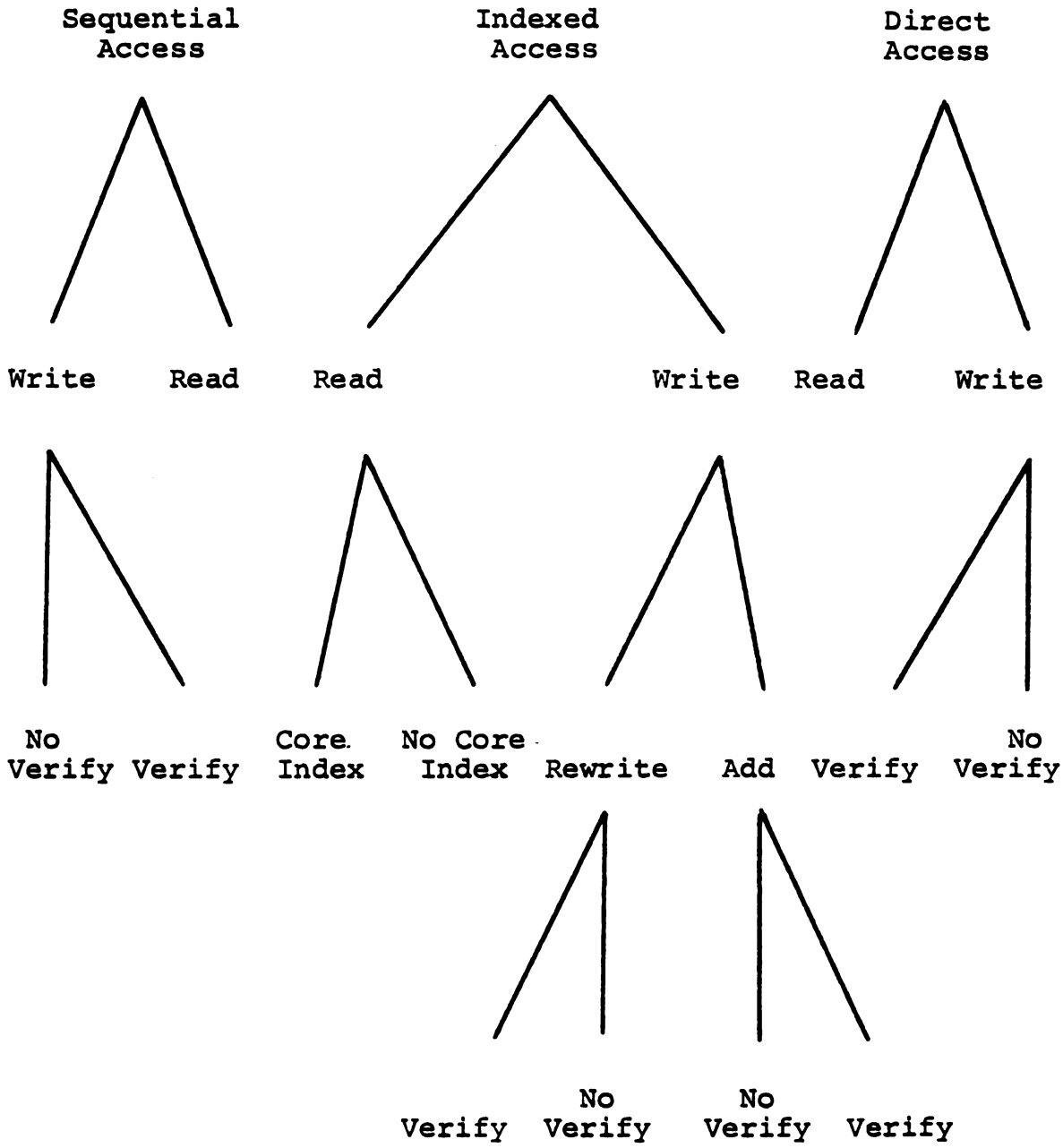

Figure 4. Disk Access Mode Diagram

Cylinder Index Read

Wait for Device	Seek Cyl. Index	Wait for Channel	Wait for Record	Wait for RPS	Search & Transfer Index
TWD	TS1	TWC	RD/2	TWR	TSC1

Track Index and Data Read

Wait for Device	Seek Track Index	Wait for Channel	Wait for Record	Wait for RPS	Search & Transfer Index
TWD	TS2	TWC	RD/2	TWR	TSC1

Wait for Channel	Wait for Record	Wait for RPS	Search & Transfer Data
TWC	RD/2	TWR	TSC2

Figure 5. Indexed Access Timing Diagram

Direct and Sequential Read

Wait for Device	Seek Cyl.	Wait for Channel	Wait for Record	Wait for RPS	Read Key & Data
TWD	TSK	TWC	RD/2	TWR	TSC

Direct and Sequential Write

Wait for Device	Seek Cyl.	Wait for Channel	Wait for Record	Wait for RPS	Write Key & Data
TWD	TSK	TWC	RD/2	TWR	TSC

Wait for Record	Wait for Channel	Read Key & Data
RD/2	TWR	TSC

Figure 6. Random Access Timing Diagram

Direct Access Disk I/O

In computing the disk file access time it will be assumed that the disk hardware uses the rotational position sensing (RPS) technology. This means that the channel will be allowed to disconnect from the disk device during both seek and search operations. A geometric distribution is assumed for the delay due to RPS.

Let BS(i,k) be the average block size for file k on device i, which is accessed by program j. Let CYL(i,k) be the size of the file and LC(i,k) its mid-point. We define the channel service time, TSC, due to access to file k on device i as

$$TSC(i,k)=2*RD/128+BS(i,k)/TR, \qquad (2.4.4)$$

where RD is the revolution time, TR is the disk transfer rate and the term 2*RD/128 is the time necessary to prepare the Rotational Position Sensing (RPS) device for data transfer. This term causes any expression in which it appears to be hardware dependent. To apply these expressions to disk devices other than IBM 3330's, this term should be modified. The I/C access rates are expressed by

$$ARAT(i,j,k)=N(i,j,k)/N(j)*ET(j). \qquad (2.4.5)$$

Then the channel utilization by device, file and program is

$$RCU(i,j,k)=ARAT(i,j,k)*TSC(i,k) \qquad (2.4.6)$$

and the total channel utilization RCU, is given by

$$RCU=sum[\,RCU(i,j,k):\ i=1,..ND;j=1,..N;k=1,..NF\,]. \qquad (2.4.7)$$

The channel wait time due to RPS is

$$TWR=RCU*RD/(1-RCU). \qquad (2.4.8)$$

To estimate the channel utilization as "seen" by program j, we have

$$RCU(j)=RCU-sum[\,RCU(i,j,k):\ i=1,..ND;k=1,..NF\,], \qquad (2.4.9)$$

and therefore we have an expected channel wait time for RPS of

$$TWR(j)=RCU(j)*RD/(1-RCU(j)). \qquad (2.4.10)$$

This is true because at the time that program j requests I/O, no other operation by program j can be in progress.

Next, the average seek time by program, file and by device is computed. First, the probability of access to each file on each device DI(i,k), is derived as follows:

$$ARAT(i,k)=sum[\,ARAT(i,j,k):\ j=1,..N\,], \qquad (2.4.11)$$

$$ARAT(i)=sum[\,ARAT(i,j,k):\ j=1,..N;k=1,..NF\,], \qquad (2.4.12)$$

and

$$DI(i,k)=ARAT(i,k)/ARAT(i). \qquad (2.4.13)$$

Disk files are considered to be organized by "cylinder", where a cylinder is the collection of all disk records which can be read or written at one physical positioning of the access mechanism or "heads". A movement of the heads from one location to another is called a "seek". Since we are accessing file k with a uniform distribution, the expected value (in cylinders) of the seek distance SK(i,k), is given by

$$E(SK(i,k))=sum[\,DI(i,l)*|LC(i,k)-LC(i,l)|:\ l\neq k\,] + DI(i,k)*((CYL(i,k)**2-1)/3*cyl(i,k)). \qquad (2.4.14)$$

This expression is based on the fact that, on the average, the seek distance between two files on the same disk will be given by the distance between their mid-points [44]. For a seek from some point within the same file the average distance will be given by the second term in the above expression. Similiary we may estimate the second moment for the seek time E(SK2(i,k)) as

$$E(SK2(i,k)) = \sum_{l \neq k} DI(i,l)*(LC(i,k)-LC(i,l))*2 + \sum_{l \neq k} DI(i,l)*((CYL(i,k)**2+CYL(i,l)**2)/12 + DI(i,k)*(CYL(i,k)**2-1)/6. \quad (2.4.15)$$

Using this expression and the previously derived mean seek distance, the variance of seek distance may be calculated using the well-known formula, $V(SK(i,k)) = E(S2(i,k)) - E(SK(i,k))**2$.

In calculating the seek time for the model, it will be assumed that the seek time is a linear function of the distance moved. This assumption is reasonable for most seeks greater than a few cylinders in distance. We define the seek time function as

$$TS=T*U(SK)+KS*SK \quad (2.4.16)$$

where SK is the number of cylinders seeked, KS is the slope, T is a constant, and U(SK) = 1 for SK greater than or equal to 1 and U(SK) = 0 for SK equal to zero. Then the seek time for an access to random file k on device i is

$$E(TS(i,k)) = (CYL(i,k)-DI(i,k))*T/CYL(i,k) + KS*(\sum_{l \neq k} DI(i,l)*|LC(i,l)-LC(i,k)|) + KS*DI(i,k)*((CYL(i,k)**2)-1)/CYL(i,k)). \quad (2.4.17)$$

Using Water's method [44], and the previous assumptions about the programs in the multiprogramming set and their

disk accesses, we calculate the variance of the seek times to random file k on device i as

$$V(TS(i,k)) = (1-DI(i,k)/CYL(i,k))*DI(i,k)*(T**2)/CYL(i,k) + 2*T*KS*DI(i,k)*E(SK(i,k))/CYL(i,k) + V(SK(i,k))*KS**2. \quad (2.4.18)$$

The wait time due to channel blocking caused by other I/O processes is given by Wilhelm's model [45] as

$$TWC(i,j,k) = sum[\, ARAT(m,l,k)*TSC(m,k)**2 : l \neq j; m \neq i \,]. \quad (2.4.19)$$

The device service time for a random read operation will be given by

$$TSD(i,j,k) = TS(i,k)+RD/2+TWR(j)+TSC(i,k)+TWC(i,j,k), \quad (2.4.20)$$

and service time for a randcm write operation will be

$$TSD(i,j,k) = TS(i,k)+RD+2*(TWR(j)+TSC(i,k))+TWC(i,j,k). \quad (2.4.21)$$

The utilization of file (i,k) due to program j will be

$$RDU(i,j,k) = ARAT(i,j,k)*TSD(i,j,k). \quad (2.4.22)$$

The respective variances for the read and write operations

are

$$V(TSD(i,j,k)) = V(TS(i,k) + (RL^{**}2)/2 + RCU(j)*(RD^{**}2)/(1-RCU(j))^{**}2 + sum[ARAT(m,l,k)*TSC(m,k)^{**}3 : l \neq j; m \neq i]/3 + TWC^{**}2, \quad (2.4.23)$$

and

$$V(TSD(i,j,k)) = V(TS(i,k)) + (RL^{**}2)/3 + 4*RCU(j)*(RD^{**}2)/(1-RCU(j))^{**}2 + sum[ARAT(m,l,k)*TSC(m,k)^{**}3 : l \neq j; m \neq i]/3 + TWC^{**}2. \quad (2.4.24)$$

From the previous computation of the variance for disk service time, we may now compute the coefficient of variation for the disk service time as

$$CS(i,j,k) = V(TSD(i,j,k))/TSD(i,j,k)^{**}2. \quad (2.4.25)$$

We finally get the disk response time by use of the queuing formula TRD(i,j,k) = TSD(i,j,k)*(1 + RHO*(CA**2 + CS**2)/2 * (1 - RDU(i,j)) where RHO = 1 + 2 * (RDU(i,j) - 1)/(CA**2 + 1), and CA and CS are the coefficients of variation of the arrival and service distributions. The term RDU(i,j) is the sum of the utilization of device i by all programs other than program j. A particular application of this formula, for Poisson arrivals (CA=1) is

$$TRD(i,j,k)=TSD(i,j,k)*(1+RDU(i,j)*(1+CS**2)/2*(1-RDU(i,j)). \quad (2.4.26)$$

To get the total response time, we must add the wait time while the disk is available and the channel is busy,

$$TWD(i,j,k)=RCU(j)*(1-RDU(i,j))*TWC(i,j,k)/2. \quad (2.4.27)$$

We now look at a slightly more complex model, a model of access for indexed data.

Indexed Disk I/O

For the indexed access method we will assume that the index is located on the same device as the data portion of the file and that a seek to this index is required. There are several cases of indexed access: (1) initial load or sequential retrieval, which can be treated using the sequential method to be given later, (2) indexed read-only, and (3) indexed read-write. We will limit the analysis of the indexed method to random reads without the index in memory (core index). Analysis of the other cases may be achieved by simple extensions of the methods used here.

We will first calculate the seek time. For a random read there are actually two seeks to be calculated; (1) the seek from the starting location of the access mechanism to the cylinder index, and (2) the seek from the cylinder index to the indexed file's data area. The expected values of

these seeks are equal to the seek times from mid-point of the file where the access arm is initially located to the mid-point of the index, and from there to the mid-point of the data area. Designating these seeks as S1 and S2 respectively, we have

$$E(S1(k,i)) = \text{sum}[DI(i,l)*|CI(i,k)-LC(i,l)|:l=1,..NF], \quad (2.4.28)$$

$$E(S2(k,i)) = |CI(i,k)-LC(i,k)|, \quad (2.4.29)$$

where CI(i,k) is the mid-point of the index for file k. Having computed the expected value of the seek distance, we may compute the expected seek times for the index and the data portions of the file as E(TS) = T + KS*E(SK) since the access mechanism will always have to move at least one cylinder. This is true because the index will always be read first and the access mechanism will never be left in the index area.

Since we actually have two I/O operations to transfer a single data record, we will compute service times for the two operations separately. First we have the channel busy time for searching and transferring the index

$$TSC1(i,k) = RD/2+2*RD/128+10/2*TR, \quad (2.4.30)$$

and the channel busy time for searching and transferring the

data record is

$$TSC2(i,k)=RD+2*RD/128+ES(i,k)/2*TR. \qquad (2.4.31)$$

Assuming independence for each segment of the timing diagram, we calculate the channel utilization for indexed reads as

$$RCU(i,j,k)=ARAT(i,j,k)*(TSC1(i,k)+TSC2(i,k))/2, \qquad (2.4.32)$$

$$RCU(j)=RCU-sum[\,RCU(i,j,k):i=1,..ND;k=1,..NF\,]. \qquad (2.4.33)$$

We then have the delay due to RPS

$$TWR(j)=RCU(j)*RD/(1-RCU(j)), \qquad (2.4.34)$$

and we calculate the channel wait time as before by Wilhelm's method:

$$TWC(i,j,k)=sum\ ARAT(m,l,k)*TSC(m,k):m\neq i;l\neq j]. \qquad (2.4.35)$$

The service time for the indexed access will be

$$TSD1(i,j,k)=TSC1(i,k)+TWR(j)+TWC(i,j,k)$$
$$+RD/2+TS(E(S1(i,k))). \qquad (2.4.36)$$

The device service time for the data access will be

$$TSD2(i,j,k)=TSC2(i,k)+2*(TWR(j)+TWC(i,j,k))$$
$$+RD/2+TS(E(S2(i,k))), \qquad (2.4.37)$$

and the rate of access to device i for all programs other than program j is

$$ARAT(i,j)=sum[ARAT(i,l,k):l\neq j;k=1,..NF]. \qquad (2.4.38)$$

The device utilization which program j finds at device i will be

$$RDU(i,j)=sum[ARAT(i,l)*(TSD1(i,l)$$
$$+TSD2(i,l))/2 : l\neq j], \qquad (2.4.39)$$

and the wait time for program j then becomes

$$TWD(j)=RCU(j)*(1-RDU(j))*TWC(j)/2. \qquad (2.4.40)$$

To complete the elaboration of the model we must find the variance of the service times. It can be shown that the variance of the distance for the initial seek, S1, is

$$V(S1(i,k))=sum[DI(i,l)*(CL(i,k)**2+CYL(i,l)**2)/12:l=1,..NF]$$
$$+sum[DI(i,l)*(1-DI(i,l))*(CI(i,k)-LC(i,l))**2:l=1,..NF]$$
$$-2*sum[DI(i,l)*|CI(i,k)-LC(i,l)|*$$
$$sum[DI(i,m)*|CI(i,k)-LC(i,m)|:m=1,..l-1]:l=1,..NF], \qquad (2.4.41)$$

where CL is the number of cylinders in the index portion of file k on device i. The seek time is then

$$V(T(S1(i,k)))=T**2+V(S1(i,k))*KS**2. \qquad (2.4.42)$$

The variance of the second seek is then calculated as

$$V(S2(i,k))=(CL(i,k)**2+CYL(i,k)**2)/12, \qquad (2.4.43)$$

and the seek time for the second seek is

$$V(T(S1(i,k)))=T**2+V(S2(i,k))*KS**2 \qquad (2.4.44)$$

The variance for the channel service time is then (RD**2)/12 and 2*(RD**2)/12 for the index read and the data read respectively. Since the variance of the rotational delay is (RD**2)/12 we have the follcwing expression for the variance of the device service time for the read of the index:

$$V(TSD1(i,j,k))=V(T(S1(i,j,k)))+V(TWC(j)) +V(TWR(j))+(RD**2)/12+V(TSC1(i,j,k)), \qquad (2.4.45)$$

where V(TWC) and V(TWR) are calculated exactly as in the case of the random access method. Similarly, we have the variance of the disk service time for the data read cpera-tion as

$$V(TSD2(i,j,k)) = V(T(S1(i,j,k))) + (RD^{**}2)/12 + 2*(V(TWC(j)) + V(TWR(j))) + V(TSC2(i,j,k)). \quad (2.4.46)$$

At this point we form the coefficient of variation exactly as before and use this in the general queuing formula to get the expected response times TRD1(i,j,k) and TRD2(i,j,k) for the indexed access method.

The extension of these methods to other forms of I/O for indexed files is straightforward. For example, to extend the previous analysis to the case where the index is in core we simply calculate the seek time component (S2) exactly as we would for a randomly accessed file. Of course the use of a core-index implies something about the memory requirements and also the CPU usage of the program. The use of a core-index makes the program a <u>different program</u> as far as the CPU and Paging submodels are concerned.

Sequential Access Disk I/O

For a sequentially accessed disk file, we calculate the timing diagram in a similar manner to calculation for the direct access case. The difference is in the calculation of the seek time. First we determine the expected value of the seek distance

$$E(SK(i,k)) = \text{sum}[\, DI(i,l) * |LC(i,k) - LC(i,l)| : l \neq k \,] + DI(i,k)/RC, \qquad (2.4.47)$$

where RC is the number of records to be read from each cyclinder of file k. The variance of the seek distance we calculate as

$$\begin{aligned} V(SK(i,k)) = {} & (1-DI(i,k)/RC) * DI(i,k)/RC \\ & + \text{sum}[\, DI(I,l) * ((CYL(i,l**2 + CYL(i,l)**2)/12 \\ & \quad + (1-DI(i,k)) * (CYL(i,k) - CYL(i,l))**2 \\ & \quad - 2*DI(i,k) * |LC(i,k) - LC(i,l)| \; : \; l \neq k \,] \qquad (2.4.48) \\ & - 2 * (\text{sum}[\, DI(i,l) * |LC(i,k) - LC(i,l)| * \\ & \text{sum}[\, DI(i,m) * |LC(i,k) - LC(i,m)| : m = 1,.l-1, m \neq k \,] : l \neq k \,]. \end{aligned}$$

We then have the expressions for the expected seek time,

$$\begin{aligned} E(TS(i,k)) = {} & (1-DI(i,k) + DI(i,k)/RC) * T \qquad (2.4.49) \\ & + K * (\text{sum}[\, DI(i,l) * |LC(i,l) - LC(i,k)| \; l \neq k \,] + DI(i,k)/RC), \end{aligned}$$

and the variance of the seek time,

$$\begin{aligned} V(TS(i,k)) = {} & 2*K*T*DI(i,k) * E(S(i,k))/CYL(i,k) + V(SK(i,k)) * K**2 \\ & + (1-DI(i,k)/CYL(i,k)) * DI(i,k) * (T**2)/CYL(i,k). \qquad (2.4.50) \end{aligned}$$

With the expected value and variance of the seek time we calculate the coefficient of variation, the disk response time, channel waiting time, and the I/O response time as we

did in the direct access case.

From the previously derived expressions for access rate by file, device and program, we can calculate the average I/O response time fcr program j by

$$XI(j)=sum[\,N(i,j,k)*TRD(i,j,k)/N(j):i=1,..ND;k=1,..NF\,]. \qquad (2.4.51)$$

I/O for Paging

The I/O response times for paging operations are computed as if the paging file were comprised of distinct subfiles for each program in the multiprogramming set. The response for each program is computed as the I/O time for random access to its paging subfile with the added condition that the paging I/O is priority scheduled using Head-of-the-Line (HOL) policy. It should be noted that the foregoing comments are specific to the implementation of the particular operating system used in this research.

Using PHI(j) as the average time to read or write a page on behalf of program j, TSD(1) as the mean service time for all I/O to the paging data set, and TSD(1,j,j) as the mean service time for access by program j to its portion of the paging file we have the following:

$$PHI(j)=TSD(1,j,j)+RDU(1,j)*TSD(1)*CK(j),$$

$$CK(j)=CK(j-1)+ARAT(1,j-1,j-1)*TSD(1,j-1,j-1), \quad (2.4.52)$$

$$CK(1)=(1+CS(1)**2)/2 \qquad j=1,..,N,$$

where ARAT and RDU have the same meanings as defined previously. All other values used in determining the response time for paging is computed exactly the same as for a random access data set.

This concludes the detailed discussion of the I/O submodel. This discussion was not intended to be exhaustive but does point out the kind of I/O models that can be used to give a more realistic treatment to I/O than is usually found in computer performance models.

Submodel Integration

We complete the model by integrating the Paging, CPU, and I/O submodels. This is accomplished by calculating the statistics for the CPU and Paging submodels, and then using these statistics as input to the I/O submodel. The I/O statistics are then used as input to the CPU and Paging submodels. This process is repeated until the model state variables converge to an equilibrium solution.

Since the model developed herein is comprised of a system of non-linear equations, the Regula Falsi, or the method of false position [30], and Aitken's Delta-Square

algorithm are used to accelerate convergence of the system. The ideas that enable the various parts of the model, the submodels, to fit together will now be examined in some detail.

The idea of a CPU cycle is extended to include program CPU execution intervals which terminate in paging operations as well as those terminating in normal (non-paging) I/O operations. Representing the average page read or write time as PHI(j) for program j, and defining the system ratio of page reads to page writes as BETA, we have the expression for the average page wait time during a paging cycle

$$ETA(j) = (1+BETA)*PHI. \qquad (2.5.1)$$

With N(j) the number of non-paging I/O operations during the execution of program j, and CTIME(j) the total problem state time during the execution of program j, we have

$$MU(j) = (N(j)+1)/CTIME(j). \qquad (2.5.2)$$

The approximate number of page fault cycles is given by

$$PG(j) = N(j)*(1/B(j)*MU(j)-1). \qquad (2.5.3)$$

We then calculate the proportion of all I/O delays due to paging as

$$PG(j)/(PG(j)+N(j)) = N(j)*(1/MU(j)*B(j)-1)/$$
$$N(j)*(1/-MU(j)*B(j)-1)+N(j) \qquad (2.5.4)$$
$$= 1-MU(j)*B(j),$$

where B(j) is the average CPU execution time per cycle for program j and PG(j) is the total number of cycles of program j's execution which terminate in page exceptions. Similarly, the proportion of cycles due to non-paging I/O operations is

$$N(j)/(PG(j)+N(j)) = MU(j)*B(j). \qquad (2.5.5)$$

Using the instantaneous page exception rate CER(j) defined previously, the number of paging cycles PG(j) is given by the product of the total number of cycles times the rate of page exception generation per cycle

$$PG(j) = (N(j)+PG(j))*CER(j)*B(j),$$

or equivalently,

$$CER(j) = PG(j)/B(j)*(N(j)+PG(j)) = 1/B(j)-MU(j). \qquad (2.5.6)$$

This implies that the CPU time per cycle B(j) is

$$B(j) = 1/(MU(j)+CER(j)) \qquad (2.5.7)$$

and B(j) can be determined. Now that we have an estimate for the average paging response time ETA(j) and the normal

I/O response XI(j), we can calculate the average overall I/O response as

$$IO(j) = MU(j)*B(j)*XI(j) + (1-MU(j)*B(j))*ETA(j). \qquad (2.5.8)$$

Turning now to a re-examination of the CPU overhead due to I/O and page management, let the normal I/O processing cycle overhead be given by ALP(j) and the average paging cycle overhead by DEL(j). The average CPU time consumed per cycle for program j will be

$$TAU(j) = B(j) + MU(j)*B(j)*ALP(j) + (1-MU(j)*B(j))*DEL(j). \qquad (2.5.9)$$

For a given program j (remember this also means priority level j), there is only one variable on the right hand side of the above expression, the variable B(j).

In taking the output of the CPU and Paging submodels as input to the I/O submodel, the device/file access rates must be disaggregated so the appropriate rates are reflected for non-paging and paging I/O (the paging I/O response is calculated by the I/O submodel just like any other I/O). The normal I/O rates are computed as MU(j)*B(j)/ET(j) and the paging rates are computed as (1-MU(j)*B(j))*(1 + BETA)/ET(j). After computing these I/O components separately, they are recombined as shown previously.

One other aspect of CPU service time to be introduced

at this point has to do with the elongation of service time due to CPU cycles being "stolen" to accomplish I/O. This effect can become significant at high I/O rates because the amount of CPU time used by the channel is proportional to the amount of data transferred and the number of I/O operations started.

This effect will be quantified by defining an "expansion factor" CYT, as the ratio of available CPU time with some level of I/O to the maximum available CPU time without I/O. This factor is expected to be different for different programs in the multiprogramming mix. Defining the amount of time stolen by the channel for an I/O operation by program j as D(j), we have

$$D(j)=DS+[(1-B*MU)*BS(j)+B*MU*BSP]*DC, \qquad (2.5.10)$$

where DS is the amount of time used by the channel tc initiate and terminate an I/O, BS(j) is the average I/O block size transferred for program j, BSP is the size of a page in the system under discussion, and DC is the amount of CPU time used by the channel per byte transferred. Fcr the present research DS = 76 microseconds, DC = 0.15 microseconds, BS = 568 bytes and BSP = 2048 bytes. These factors are all hardware dependent.

Then the amount of CPU time stolen each second for I/O on behalf of program j is given by D(j)/ET(j) so the

fraction of CPU cycles available for instruction execution is 1-D(j)/ET(j). A program of higher priority than program j must be in the I/O stage if program j's instruction are executing, so the fraction cf CPU time available due to the higher priority program (k) is 1-LAM(k)*D(k). Combining this with the results for a lower priority program, it can be shown that

$$CYT(j)=\text{prod}[\,1-LAM(k)*D(k):k=1,..j-1\,] \qquad (2.5.11)$$
$$*\text{prod}[\,1-D(k)/ET(k):k=j+1,..,N\,].$$

CYT may be computed in sequential fashion in the CPU submodel by

$$CYT(j)=CYT(j-1)*(1-LAM(j-1)*D(j-1))/(1-D(j)/ET(j)),$$
$$CYT(1)=\text{prod}[\,1-D(k)/ET(k):k=2,..,N\,], \qquad (2.5.12)$$
$$\text{for } j=2,3,..,N.$$

The expected length of a CPU execution interval, whether it is B, TAU, ALP or DEL may be elongated simply by dividing by the corresponding CYT.

This completes the submodel integration. At this point all of the essential elements of the model have been put together. The conditions for convergence to an equilibrium and the extent to which these conditions are met by the present model will now be examined.

Model Convergence

If we represent a cycle of iterations of the CPU, Paging, and I/O submodels by the real valued functions f(.), defined over Euclidean N-space, we have

$$f(j)=LAM(j)-LAM'(j) \quad \text{for } j=1,..N, \qquad (2.6.1)$$

where LAM(j)=1/IO(j) and LAM'(j)=1/IO'(j) are the outputs of the I/O submodel (and consequently input to the CPU and Paging submodel) on successive iterations. LAM and LAM' are interpreted as instantaneous I/O response rates. The equilibrium condition requires that

$$f(j)=0 \quad \text{for } j=1,..N. \qquad (2.6.2)$$

Since the model is a system of non-linear equations, and the iterative method of alternating between CPU-Paging submodel and I/O submodel calculations produces an oscillating series, we have employed a variation of Newton's Method, the Regula Falsi [30], to accelerate convergence. The algorithm is based on the general form

$$g(j)=x(j)-f(j)/f'(j) \quad j=1,..N, \qquad (2.6.3)$$

where x(j)=LAM'(j) is a scalar and f'(.) is the derivative of f(.) with respect to x(j). Representing the vector forms for f, g, y, and x by the upper cases F, G, Y, and X we

have the vector condition for convergence of the algorithm:

$$||G(X)-G(Y)||\leq M*||X-Y||, \qquad (2.6.4)$$
$$\text{for} \quad ||X-Y||<RHC$$

where ||.|| is a norm, M is a scalar less than one, and the vectors X and Y are in a sphere of diameter RHO about the equilibrium point, EP, in NP-space [37].

We are assured of the existence of the point EP by the Mean Value Theorem, since each f(j) is continuous in any neighborhood of EP(j) and takes on both positive and negative values in this neighborhood. Because of the complexity of the full expression for f(.), it is very difficult to manipulate the derivative symbolically --although it is possible to use the same algorithm which calculates f(.) to calculate f'(.) numerically.

If we represent the k-th iteration of the vector X by X(k), we can express equation 2.6.3 as

$$g(j;X(k))-g(j;X(k-1))=-f(j;X(k))/f'(j;X(k)) \qquad (2.6.5)$$
$$\text{for } j=1,..N \text{ and } k=1,2,...$$

With this expression in mind, we will take a closer look at the function f(.). To reduce the complexity of the analysis, we will simplify the model in a way which will not interfere with the generality of the results.

First, we will assume that all block sizes are equal and that all file accesses are to disk. Furthermore we will limit the access method to direct (or random) accesses uniformly distributed over a single disk module. This is really not as great a simplification as it seems since it is many times more complicated to calculate response times for many programs accessing a single disk drive than it is to compute the response time for each program accessing a unique disk drive. The latter case involves no iteration since each program's I/O response will be almost totally independent of every other program's I/O response, and therefore easier to compute (this is because there will be no queuing for devices).

The foregoing assumptions will ensure that the service time (TSD) for every program will be identical. Ignoring the channel service time, and designating the disk service time by TSD, we have some natural constraints to work with. For example, the access rate for all programs must be less than 1/TSD since a device cannot be utilized more than 100%. Representing the instantaneous access rate for program j by x(j), an analysis of a single iteration of the algorithm (the function f), we have that $f(j;x(j)=0) < 0$ and $f(j;x(j)=1/TSD) > 0$ for every program j. Furthermore, it can be shown that $f(j;x(j)) \leq f(j+1;x(j+1))$.

We can guarantee that $f(N\ x(N)=0) < 0$ by the imposition of the requirement that the utilization of the disk be

less than or equal to 100%. Given the access rates ARAT(l), and a single disk device, we have

$$
\begin{aligned}
& f(j;x(j)) = x(j) + \text{sum}[\,ARAT(l) : l \neq j\,] - 1/TSD, \\
& \text{sum}[\,ARAT(l) * TSD : l = 1,..N-1\,] < 1, \\
\Rightarrow \quad & \text{sum}[\,ARAT(l) : l = 1,..N-1\,] < 1/TSD, \qquad (2.6.6) \\
\Rightarrow \quad & f(j;x(j)=0) = \text{sum}[\,ARAT(l) : l = 1,..N-1\,] - 1/TSD < 0.
\end{aligned}
$$

For $x(j) = 1/TSD$ we have

$$
\begin{aligned}
& f(j;x(j) = 1/TSD) = 1/TSD \\
& \quad + \text{sum}[\,ARAT(l) : l = 2,..N\,] - 1/TSD \qquad (2.6.7) \\
& \quad = \text{sum}[\,ARAT(l) : l = 2,..N\,] \geq 0,
\end{aligned}
$$

since each rate ARAT(l) is strictly positive and converges to zero as x(j) becomes infinitely large.

To show that $f(j\ x(j)) \leq f(j+1;x(j+1))$ we have

$$
\begin{aligned}
f(j;x(j)) = {} & f(j+1;x(j+1)) - (x(j)^{**}2) * (C(j) + x(j) * \\
& (W(j) + IO(j)) * ARAT(j) * ARAT(j+1). \qquad (2.6.8)
\end{aligned}
$$

Since all of the variables following the negative sign in the above expression are greater than or equal to zero, the necessary condition prevails. Because the functions involved are continuous in the domain of interest, we are assured of a solution.

It can also be shown that the functions f(.) are differentiable and that the partial derivatives of f(j) with respect to x(j)--Df(j)/dx(j)--are ordered

$$Df(1)/dx(1) \leq Df(2)/dx(2) \leq \ldots \leq Df(N)/dx(N) = 1, \qquad (2.6.9)$$

and the second partials DDf(j)/dx(j) are non-negative everywhere,

$$DDf(1)/dx(1) \geq DDf(2)/dx(2) \geq \ldots \geq DDf(N)/dx(N) = 0. \qquad (2.6.10)$$

These conditions assure us of a unique solution and they also assure us that the derivative of each function is greater than zero and less than or equal to one (1) in the neighborhood of the solution (the derivatives are very nearly equal to one in this neighborhood). We have that

$$Df(1:EP(j))/dx(1) > TSD*(f(1:1/TSD) - f(1:0)) \qquad (2.6.11)$$

$$|g(j:x(j)) - g(j:y(j))| = |x(j) - y(j) + f(j:x(j))/f'(j:x(j)) - f(j:y(j))/f'(j:y(j))|. \qquad (2.6.12)$$

By Taylor's Theorem,

$$f(j:z(j)) = f(j:EP(j)) + f'(j:EP(j))*(z(j) - EP(j)), \qquad (2.6.13)$$

implying that

$$|g(j:x(j))-g(j:x(j))|=|x(j)-y(j)+CHI*(y(j)-x(j))|$$
$$=|1-CHI|*|x(j)-y(j)|, \qquad (2.6.14)$$

where CHI is a non-negative scalar which is less than one. Using $||X|| = \max[x(j):j=1,..N]$ as the definition of the norm, we chose RHO so that $||X-EP|| < RHO$ implies by continuity that

$$f'(j:EP(j))*4/5<f'(j:x(j))<f'(j:EP(j))*5/4,$$

implying that

$$4/5<f'(j:EP(j))/f'(j:x(j))<5/4. \qquad (2.6.15)$$

This means that $4/5 < CHI < 5/4$ or $-1/4 < 1-CHI < 1/5$, therefore

$$|1-CHI|< 1/4. \qquad (2.6.16)$$

Substituting this into the previously derived expression, we have

$$||G(X)-G(Y)||\leq|1-CHI|*||X-Y||<(1/4)*||X-Y||. \qquad (2.6.17)$$

With this we conclude the demonstration that the use of Newton's method is warranted and that the algorithm converges to a unique solution. To accelerate the rate of convergence of this process, we apply Aitken's Delta Square algorithm to estimate a new point from three previous estimates by Newtons method. The task now is to develop the

experimental apparatus to validate the model and determine its usability.

III. MODEL VALIDATION AND EXPERIMENTAL DESIGN

Experimental Plan

The computer runs for estimation and validation were performed on an IBM System/370 Model 148 in the Detroit Datacenter. The IBM OS/VS1 operating system and the IBM software products, Systems Measurement Facility (SMF), and OS/VS1 Utilization Monitor were used to collect and manipulate performance data. Making the experimental runs on the same computer as the base runs does not detract from the results since the programs in the base runs and the experimental runs were in totally different combinations and with different priorities. This is, after all what the model is supposed to predict. The only additional information that could have been gained by executing the experimental runs on a different computer from the base runs is the extent to which execution timings on different model computers are not proportional. For example the ratio of execution timings for scientific and commercial instruction mixes on two different models of the same computer "family" will be different. The model developed here does not attempt to give an answer to this problem and a decision was made to use published instruction execution rates for an "average" commercial job mix as is the common practice.

To a certain extent the proposed model is dependent upon the particular implementation of the operating system used. The implementation of the operating system used in making the measurements is interrupt driven priority-resume dispatching for batch programs. This research was conducted using only one operating system but this does not seriously limit the generality of conclusions reached because differences within the class of operating systems defined above are differences in parameter values and furthermore this dispatcher is representative of the majority of operating systems in current use. Examples of these parameters are the quanta of CPU overhead for I/O interrupt, start I/O, dispatch processing, and paging overhead.

Estimates were obtained for the paging submodel coefficients which correspond to the page-out/page-in ratios (BETA), working set allocation parameter (L), and the paging indices (A). The estimation began with the execution of a selected workload and the measurement of its performance. The measurement data was then fitted to the model by use of multivariate regression analysis. Finn's MULTIVARIANCE program was used for estimation of the regression coefficients [22].

Performance measurement was achieved by means of software monitors and job accounting data collection. The response data collected included program CPU usage, total CPU usage, channel busy counts, device busy counts, paging

and other I/O counts and working set sizes. These data were manipulated to give the performance values of interest: total CPU utilization, program CPU utilization, channel utilization, average working set sizes, paging rates, program I/O operations. Further transformations were made to estimate each program's page index (A), mean CPU service time (1/MU), and mean I/O response time (IO).

To complete the experimental plan, the apparatus employed, and the programs used in parameter estimation and model validation are described. Next, a description is given of the instruments or measurement software and data reduction programs used, and the measures to which they relate. The last section of the chapter provides descriptions of the experimental design for estimation, the design for validation, and the respective hypotheses tested.

Programs for Measurement and Control

The programs used in this research were of three types (1) those written by the experimenter to predict program behavior and estimate parameters, (2) those designed and programmed by the experimenter so that the experimental variables could be controlled in model validation, and (3) statistical programs. The programs used for prediction are basically the model developed in chapter two. This model was programmed in APL [31] and executed on an IBM System/370. The experimental programs were written in COBOL and the statistical programs used in this research were

Finn's MULTIVARIANCE package [22].

The parameter estimation workload is composed of 8 programs randomly chosen from a population consisting of 8 "CPU bound" programs and 8 "I/O bound" programs. The definition for "I/O bound" used here is that, for a program run in isolation, greater than 75% of the elapsed time would be spent waiting for I/O operations. Likewise "CPU bound", as used here, means that the program would consume more than 50% of the available CPU cycles during its execution.

The programs used in the model validation were "synthetic" programs [10,42] specially constructed to vary over the cells of the experimental design. The synthetic programs were written as extensions to the synthetic programs mentioned earlier in this thesis but they were modified to enable the paging index A(j) to be varied. More will be said about this in the discussion of experimental design.

Experimental Variables

There are 5 independent variables and 7 dependent variables in the experiment. The program related independent variables are: (1) the page index, A(j), and (2) the variance of CPU service time VS(j). Three other independent variables are environment related and consist of: (1) The variance of I/O service time VI(j), (2) the priority of the program being measured, j, and (3) the total number of

programs in the multiprogramming mix, N. A decision was made to eliminate the memory variable R, because its effect is confounded with the factor N, the number of programs.

The dependent variables are: (1) problem program CPU utilization CP(j), (2) program elapsed cycle time ET(j), (3) total program paging rate PR(j), (4) program page-in rate PI(j), (5) program channel utilization RCU(j), (6) total system overhead OHT(N), and (7) average program working set size w(j).

The system parameters are: (1) CPU overhead for normal I/O, ALP(j), (2) CPU overhead for Paging, DEL(j), (3) the ratio of page-writes to page-reads, BETA(j), and (4) the working set allocation weights, L(j).

Instruments and Measures

The instruments used in the collection of elapsed times, problem state time, wait time, paging counts and working set size samples by program were the job accounting facility of the IBM OS/VS1 operating sytem, the Systems Measurement Facility (SMF), and the IBM proprietary program, OS/VS1 Utilization Monitor. The channel and device utilizations and working set sizes are based on sampling. System paging and other I/O rates are based on counts and CPU times are based on actual measurements.

In other cases the variables in this experiment were not directly observable, so indirect measures were

constructed. While it is fairly easy to estimate the problem state CPU utilization CP(j) as the ratio of total problem state time to elapsed time, the overhead due to a specific program is not generally attributed to that program by job accounting routines or program monitors. Because of this, estimates of COH(j) must be achieved by a partition of the total system overhead. The total system overhead is an estimate based on the difference between the elapsed time and the wait time and problem state time or it is based on sampling by performance monitors.

In order to estimate the parameters ALP(j) and DEL(j), estimates must be obtained for COH(j), but it has already been stated that COH(j) is not measurable. One way to partition total system overhead for each program is to assume that the ratio of problem state CPU utilization to overhead is the same for each program as new programs are added to the multiprogramming mix. Suppose that we have L programs in the mix and CP(j) and COH(j) for j=1,..,L. We then add program K+1 and get CPT, the total CPU utilization, and the program utilizations CP'(j) for j=1,..,K+1. The above assumption implies that

$$COH'(j)=COH(j)*CP'(j)/CP(j), \quad \text{for } j=1,..,K \qquad (3.4.1)$$

$$COH(j)/CP(j) = CER(j)*DEL(j) + MU(j)*ALP(j) \qquad (3.4.2)$$
$$= CER'(j)*DEL(j) + MU(j)*ALP(j)$$

$$w(j) = w'(j) \quad \text{for } j=1,..,K. \qquad (3.4.3)$$

Thus the above assumptions imply that the working sets for program j is identical in the runs with K and K+1 programs . This is only possible if memory is not a binding constraint.

Since we have neither a direct nor an indirect means of estimating COH(j), and since COH(j) is a function of ALP and DEL-- both of which must also be estimated--some other means of partitioning the total system overhead had tc be devised. It was then hypothesized that ALP and DEL increase with increasing j (lower priority) since the dispatcher must process more dispatch queue entries before arriving at the last queue entry. It is also likely that ALP and DEI both increase with increasing N since more lists and tables must be checked by the operating system at every interrupt. Finally, it was hypothesized that the amount of processing that must be accomplished by the operating system before an I/O or paging operation can be sucessfully initiated is proportional to the depth of the I/O queues, and the depth of the I/O queues is proportional to the the aggregate rate of I/O operations, T(j). For the overhead variables we have

$$ALP(j,N,T(j)) = K1 + j*K2 + N*K3 + T(j)*K4, \qquad (3.4.4)$$

and

$$DEL(j,N,T(j)) = L1 + j*L2 + N*L3 + T(j)*L4, \qquad (3.4.5)$$

where $T(j) = sum[\, 1/ET(l) : l=1,..N;\ l \neq j\,]$. We then have for program j's overhead

$$COH(j)/CP(j) = ((1-B(j)*MU(j))*DEL(j,N,T(j)) + B(j)*MU(j)*ALP(j,N,T(j))/B(j), \qquad (3.4.6)$$

or

$$COH(j) = (1-B(j)*MU(j))*DEL(j,N,T(j))/ET(j) + B(j)*MU(j)*ALP(j,N,T(j))/ET(j). \qquad (3.4.7)$$

We now assume that there is insignificant paging ($B(j)*MU(j) = 1$) and sum the individual program overhead terms giving

$$OHT(N) = X(N)*K1 + Y(N)*K2 + N*X(N)*K3 + (X(N)**2 - S(N))*K4 \quad \text{for } N=1,..,7, \qquad (3.4.8)$$

where $X(N) = sum[\, 1/ET(j) : j=1,..N\,]$, $Y(N) = sum[\, j/ET(j) : j=1,..N\,]$, and $S(N) = sum[\, 1/ET(j)**2 : j=1,..N\,]$. We have a system of 7 equations in 4 unknowns for which a solution exists if the matrix of coefficients times its transpose is non-singular. This will be the case if no column or row is equal tc a combination of other columns or rows. The condition that there be negligible paging can be guaranteed by manipulation of the large real memory of the experimental system.

Executing the ALP estimation workload in a paging environment provided the basis for computing the overhead caused by paging as the residue of the CPU overhead due to I/O which was estimated with the ALP coefficients. For the residue due to paging we have

$$OHT'(N)=X(N)*L1+Y(N)*L2+N*X(N)*L3+(X(N)**2-S(N))*L4$$
$$\text{for } N=4,..,8. \qquad (3.4.9)$$

The first 3 runs were eliminated because there was no paging for N<4 since the size of each program was identical and the size of main memory was constrained to the size of three copies of the program. From two replications of these 5 experiments, equations were derived to estimate the 4 paging coefficients. The overhead expressions were then combined to get the overhead for the "mixed" model.

To estimate the ratio of page-writes (PO(j)) to page-reads (PG(j)), consider the conditions which result in a page-write operation. A page exception will result in a page-write operation if there are no page frames that are unreferenced and there are no referenced pages that have not been modified. The probability of this event--BETA(j,N)--times the number of trials (PG(j)) will give the number of pages written or PO(j) = BETA(j,N)*PG(j). It was thought that BETA would be a function of the amount of real memory in the system and the amount of memory demanded by all the programs in the system and not depend on

program j alone, i.e. a system parameter.

The experiment was carried out with printer output spooled but not printed until after program termination because of the need to reduce the complexity introduced by spooling and measuring additional programs. This does not reduce the usefulness of the model because each execution of a spool task can be considered to be the execution of a separate program which requires a printer and "spooled" print output from some other program.

A spool program is just like any other in the model except that its priority will be set higher than that of a normal program. In this sense the model would allow the priority to "float" according to the number of system programs (spool tasks) which are active.

Experimental Design

Design for Parameter Estimation

The design for parameter estimation consisted of 16 runs of one to 8 programs for the estimation of coefficients for ALP, and 10 runs for the estimation of coefficients for DEL, L and BETA (see Figure 7). The hypotheses tested were the following:

1. Larger values for the program I/O overhead parameter (ALP) are expected for programs executing at lower priority levels (larger values of j)
2. Larger values of the program I/O overhead parameter

(ALP) are expected for programs executing in larger multiprogramming sets (N)

3. Larger values of the program I/O overhead parameter (ALP) are expected with larger system I/O access rates (T(j))
4. Larger values of the paging overhead parameter (CEL) are expected for programs executing at lower priority levels (larger j)
5. Larger values of the paging overhead parameter (CEL) are expected for programs executing in larger multiprogramming sets (N)
6. Larger values of the paging overhead parameter (CEL) are expected with larger system I/O access rates (T(j))
7. The page-write to page-read ratios BETA are expected to be identical for programs executing concurrently
8. The page-write to page-read ratio BETA is expected to be larger when a program is executed in a larger multiprogramming set than it is when the program is executed in a smaller multiprogramming set at the same priority
9. The overall system page-write to page-read ratio BETA is expected to be larger for larger multiprogramming sets
10. The page allocation weights (L(j)) for all programs executing concurrently are expected to be identical

Number of Programs---	1	2	3	4	5	6	7
Priority							
1	PGM1	PGM1	PGM1	PGM1	PGM1	PGM1	PGM1
2		PGM2	PGM2	PGM2	PGM2	PGM2	PGM2
3			PGM3	PGM3	PGM3	PGM3	PGM3
4				PGM4	PGM4	PGM4	PGM4
5					PGM5	PGM5	PGM5
6						PGM6	PGM6
7							PGM7

Note: PGMx is the x-th program randomly chosen.

Figure 7. Experimental Design for Estimation

This concludes the discussion of experimental design for estimation.

Experimental Design for Validation.

The objective of the experiment was the determination of the limits of the model and the extent of its validity. The five independent variables were varied over high and low values to form a fixed crossed factorial design with 32 cells. Defining a low number of tasks as 6 and a high number of tasks as 7, the experiment was effected by 16 computer runs of 6 programs each and 16 computer runs of 7 programs each for a total of 32 computer runs.

The experiment used 4 disk drives in addition to the "system" disks. The first disk was accessed by all programs for paging and the third disk was accessed by the experimental program only. The second and fourth disks were only accessed by the control programs (CNTL).

The criterion variables for this design were relative errors consisting of the differences between the predicted values of the variables and the experimental or measured values divided by the experimental or measured values. Thus a positive value of the transformed variable signifies over-prediction (positive error) and a negative value signifies under-prediction (negative error). Using a linear model to test hypotheses by means of the analysis of variance, multivariate tests were performed for each of the

independent variables or effects.

To allow for sufficient degrees of freedom to estimate the error term, at least two replications were required in each cell of the design. Since the number of computer runs involved would have been quite large (64), a decision was made to use a 1/2 or "fractional" replicate, allowing the estimation of all main effects as well as two factor interactions [16,37]. This design reduced the size of the experiment to 2 replications of 16 runs (see Figure 8). The experimental data was tested using the Analysis of Variance routines of Finn's MULTIVARIANCE program [22].

1CCCCC 00000	1CCCCCC 00001	CCC1CC 00010	CCC1CCC 00011
2CCCCC 00100	2CCCCCC 00101	CCC2CCC 00110	CCC2CCC 00111
3CCCCC 01000	3CCCCCC 01001	CCC3CC 01010	CCC3CCC 01011
4CCCCC 01100	4CCCCCC 01101	CCC4CC 01110	CCC4CCC 01111
5CCCCC 10000	5CCCCCC 10001	CCC5CC 10010	CCC5CCC 10011
6CCCCC 10100	6CCCCCC 10101	CCC6CC 10110	CCC6CCC 10111
7CCCCC 11000	7CCCCCC 11001	CCC7CC 11010	CCC7CCC 11011
8CCCCC 11100	8CCCCCC 11101	CCC8CC 11110	CCC8CCC 11111

Note: Numbers 1-8 represent "Synthetic" programs, and "C" represents the control program. The underlined cell identifications are the cells which must be included in a "half-replicate" fractional design.

Figure 8. Experimental Design for Validation

The five-way factorial design for the experiment was chosen to permit testing of the following null hypotheses:

1. The mean relative error in the prediction of the experimental outcomes is less than or equal to .15
2. The mean relative error in the prediction of the experimental outcomes due to variation in page index (A) is less than or equal to .15
3. The mean relative error in the prediction of the experimental outcomes due to variation in CPU service variance (VS) is less than or equal to .15
4. The mean relative error in the prediction of the experimental outcomes due to variation in I/O service variance (VI) is less than or equal to .15
5. The mean relative error in the prediction of the experimental outcomes due to variation in multiprogramming level (N) is less than or equal to .15
6. The mean relative error in the prediction of the experimental outcomes due to variation in priority level (j) is less than or equal to .15

Before testing the above hypotheses, tests of all 10 two-factor interactions were planned.

One characteristic of the Analysis of Variance is that tests of main and fixed effects are only meaningful in the absence of significant interactions. The step-wise strategy for significance testing is to test interactions first and if the null hypothesis is maintained, to continue testing

the fixed effects in reverse order. If a significant interaction is found, main and fixed effects cannot be tested because they are "confounded" with the interactions.

A partial solution to this dilemma is to construct confidence intervals about the means of the criterion variables. This will give some information about errors but will not answer any questions about sources of error. The following null hypotheses were tested by means of 95% confidence intervals:

7. | Relative error predicting CP | ≤ 0.15
8. | Relative error predicting ET | ≤ 0.15
9. | Relative error predicting PR | ≤ 0.15
10. | Relative error predicting PI | ≤ 0.15
11. | Relative error predicting RCU| ≤ 0.15
12. | Relative error predicting OHT| ≤ 0.15
13. | Relative error predicting w | ≤ 0.15

For significance testing, the independent variables were arranged in the order in which the largest errors could be predicted based on taking "approximate" derivatives of the dependent variables with respect to the independent variables. The basis for this assertion is that an error in measurement of an independent variable or in estimation of a parameter may be considered to be a perturbation of the variable or parameter. If it is assumed that the structure of the model is an adequate representation of the phenomena under investigation, the effect of the perturbation may be

viewed as a change in the dependent variable, i.e. a partial derivative.

An example of this technique is the effect on working set size of a change in the variance of the CPU service time (for Erlang-1 service),

$$Dw(j)/dVS(j)=w(j)*(1-w(j)/R)*(1-C(j)/ET(j))/4*VS(j). \quad (3.5.1)$$

The effect on working set size of a change in real memory when memory constraints are active (sum[S(j):j=1,..N] > R) is given by

$$Dw(j)/dR=w(j)/R. \quad (3.5.2)$$

This type of "approximate" derivative indicated that the appropriate order of the variables for step-down and step-wise testing was: w, PI, PR, ET, CP, COH, and RCU for the dependent variables, and A, VS, VI, N, and j for the independent variables.

A feature of the experimental design is that each run was a measurement of only one program, with the other programs in the multiprogramming set controlling the environment. The program being measured ran in a "matrix" of copies of a "control" program. The control program was selected to exhibit "average" behavior compared to the

experimental programs. A table of level values for each independent variable, and a table of levels for each of the synthetic programs is given in Figure 9.

PROGRAM-RELATED VARIABLES

Synthetic Program Number	Code	Paging Activity Index	CPU Service Variance	I/O Response Variance	Expected Error
1	000	low	high	high	low
2	001	low	high	low	
3	010	low	low	high	
4	011	low	low	low	
5	100	high	high	high	
6	101	high	high	low	
7	110	high	low	high	
8	111	high	low	low	high

ENVIRONMENTAL VARIABLES

Code	Pageable Memory	Priority	Number of Programs	Expected Error
000	fixed	low	low	low
001	fixed	low	high	
010	fixed	high	low	
011	fixed	high	high	
100	fixed	low	low	
101	fixed	low	high	
110	fixed	high	low	
111	fixed	high	high	high

Figure 9. Levels of Independent Variables

The Synthetic Program

At this point a discussion of the programs used in the experimental manipulations is in order. The experimental programs are COBOL programs which process a sequential file of "transactions" from disk against a direct update file on disk. The program is written so that it can be run any number of times against the files. Control information is provided by means of execution parameters on a control card. These parameters tell the program how large the records on the disk are, how many records there are in each file, how many passes to make through the transaction file, the average number of times to execute the "compute kernal" between I/O's, the interval of variation to use in computing a random number of kernal executions on each transaction record and the amount of variation to use in accessing its own instructions in memory for paging.

Variation of the I/O is provided by the placement of the data sets, the size of the direct files, and the record sizes that are built when the files are created. The files are created by a separate program which creates the direct file sequentially then creates the sequential file by generating random relative record numbers of records in the direct access file. In practice, the variation in I/O response was controlled by data set placement and record sizes were held constant.

The compute kernal of the program selected 10 numbers

from an array of 1000 numbers and performed non-trivial arithmetic operations on them which were self-checking (they had to match previously computed numbers in the transaction file). This turned out to be so compute-bound that the number of compute passes between I/O had to be held tc 2. The actual number cf compute passes could be held to exactly two by selecting a variation of 0 or the maximum variation could by generated by selecting 2, causing the program to randomly choose an equally likely integer in the interval [0,4].

The program's executable code (as well as the 1000 entry numeric array) was duplicated five times. Each section of code was sufficiently different that a paging parameter of 0 would result in sequential execution of each block of code processing the data in corresponding data blocks. A paging parameter of 1 caused the program to skip one-half of the code and data in each block. A paging parameter of 2 caused the program to skip one-fourth of the code and data in each block. This particular scheme was chosen because it was felt that sequential execution would result in the lowest paging rates since fewer pages would be referenced during a specific execution interval.

The parameter levels chosen for the control programs were 1 for paging index, 1 for CPU variance, and 2 fcr number of kernal passes. The I/O was determined by making the files 7 and 2 cylinders in size and locating their

centers 4 cylinders apart on the same phsysical disk drives. The I/O for the experimental programs was controlled by making the files 9 and 3 cylinders in size and locating their centers 3 and 15 cylinders apart for the high variance and the low variance versions of the program respectively.

As in parameterization, the experimental data were collected by SMF and OS/VS1 Utilization Mcnitor and reduced for input (along with the predicted results) to the MULTI-VARIANCE program.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

Parameter Estimation

Normal I/O overhead

Hypotheses 1, 2, and 3. Two replications of these runs were made and the measured and calculated results are listed in appendix A, Tables A1-A4. The data were analyzed using linear regression on the model equation 3.4.4. Initial analysis yielded a significant contribution to variance for all but the last factor. The aggregate file I/O access rate contributed to an increase in ALP but the increase was not significant. This term was subsequently dropped from the model and the data re-analyzed. The statistics for the revised regression analysis is given in Table 1. The statistics in Table 1 supports hypotheses 1 at the .0001 level. The effect of multiprogramming set (N) is significant at the .0001 level but its direction is contradictory to hypothesis 3. Adjustment of the predicted mean sc that it is equal to the observed mean yields a positive intercept. This reflects the fact that in a lightly loaded system, the page management routines use idle CPU time to search page tables and maintain page queues.

Table 1. I/O Overhead Regression Analysis

SOURCE	RAW REGR COEFFS	STD ERROR OF EST
CONSTANT	2.273370E-03	8.077765E-05
PRIORITY (j)	3.839576E-04	3.428922E-05
MPG SET (N)	-2.854969E-05	4.893795E-06

STD DEV OF DEP VARIABLE (OH) = 0.0212

MULT-R-SQUARED=.9994 F(3,12)=6776.648 P<.0001

STEP-WISE REGRESSION

CONSTANT	F(1,14)= 753.360	98.18% ADDL VAR	P<.0001
PRIORITY	F(1,13)= 91.806	1.60% ADDL VAR	P<.0001
MPG SET	F(1,12)= 34.034	0.17% ADDL VAR	P<.0001

ADJUSTMENT TO MEAN = 0.008475

Paging I/O overhead

The same programs that were run in a non-memory constrained environment for ALP estimation were then run with pageable memory constrained to 312K. This amount of memory was deemed appropriate to run three of the programs (each with total memory requirement of 162K) with negligible paging. The paging experiments were started with 4 programs in the mix and another program was added with each succeeding run up to a final run with 8 programs. In the final run the system was under such stress that the monitor data that was being logged was incomplete and it was not possible to get measurements on two of the programs in the mix. When the sequence of runs was repeated, the replication with 8 programs was therefore omitted.

The data collected from these runs is reflected in appendix A, Tables A5-A8. Equation 3.4.4 was then used with the coefficients for ALP to predict the overhead due to non-paging I/O. The resultant overhead was subtracted from the measured overhead to form the residue. The data were then analyzed using linear regression with the I/O rates in equation 3.4.5 replaced by the paging rates. A surprising aspect of this process was that the overhead with paging was actually lower than it was without paging at smaller multiprogramming sets (N). Analysis of these runs (Table 2) shows that the intercept for paging overhead is negative.

An interpretation for this anomaly is that the page management routines use CPU idle time to perform housekeeping functions and look for work to do even in the absence of paging. When the CPU is highly utilized and no paging is taking place this discretionary work is effectively bypassed. However when paging is taking place this work cannot be bypassed and much of it is performed as a part of normal page I/O processing. Ideally the intercepts for ALP and DEL should cancel each other since no overhead should be expended when I/O is not being done.

Hypotheses 4, 5, and 6. Initial statistics for analysis of the model equation 3.4.5 indicated that the independent variables for priority level j, multiprogramming set N, and the aggregate paging rate T(j), all had effects in the hypothesized direction on the paging overhead parameter DEL.

Table 2. Paging Overhead Regressicn Analysis

SOURCE	RAW REGR COEFFS	STD ERROR OF EST
CONSTANT	6.165776E-03	6.783070E-05

MULT-R-SQUARED=.9993 F(1,6) =8262.718 P<.0001

ADJUSTMENT FOR MEANS=-0.0129923

However the effects were not statistically significant since the significance levels for these effects were .79, .99, and .40 respectively.

The constant term was left as the only term in equation 3.4.5, making paging overhead directly proportional to paging rate. The data were re-analyzed using only the constant factor in equation 3.4.5, yielding the coefficients in Table 2.

Paging Ratio Estimation

Since paging was needed to perform this estimation the same set of runs were used for both DEL estimation and paging ratios (BETA) estimation. An immediate source of difficulty is the fact that page-out counts recorded by most operating systems are not the same as those used in this formulation. The theory predicts the number of pages that must be written out to accomodate pages that are read for a specific program. The identification of the program which is losing the page is of no concern. However job acccunting routines do not record the identification of the program

which caused the page to be written but only adds to a counter of pages lost for the program losing the page.

There are two alternatives to this dilemma. Hypothesis 7 could be assumed and hypothesis 9 tested, or it can be assumed that the number of pages written on behalf of a program is equivalent to the number of pages which that program causes to be written and proceed with the analysis. Proceeding on the latter assumption, results of the hypothesis tests must be interpreted in view of the above discussion.

Hypotheses 7 and 8. Hypotheses 7 and 8 were tested by performing a 4X4 factorial analysis of variance on the PO/PI ratios of the same 4 programs for multiprogramming sets N = 4, 5, 6 and 7. The within-group sum of squares was used as the error term. Results of this analysis are summarized in Table 3. Since there are no significant interactions main effects may be tested. Hypothesis 7 was not supported since there was a priority effect at the .0009 significance level. The multiprogramming effect was positive (consistent with hypothesis 8) but the significance level was .124. The cell means predicted from the coefficients in Table 3 were used as the BETA(j,N) for the predictive model.

Hypothesis 9. Hypotheses 9 was tested by performing a one-way analysis of variance on the system-wide paging ratios computed from the paging estimation runs with 4, 5, 6 and 7 programs in the multiprogramming sets, yielding 8

samples. From observations of cell means it was decided to test the hypothesis using orthoganal polynomial contrasts. Hypothesis 9 was not supported at the higher multiprogramming levels since a quadratic effect was found at the .0004 significance level. The Analysis of Variance table for this analysis is given in Table 4.

Page Allocation Weights

This factor tests the Paging submodel's accuracy in predicting working set allocations. The measure for this factor L, is derived from the Kuhn-Tucker conditions for minimization of the total paging rate where program j's paging rate is weighted by L(j). Summing the L(j)s to 1 and assuming that paging indices A(j) are identical, we have the following estimator for L(j):

$$L(j,N) = (w(j)^{**}2)/CPP(j) * sum[(w(l)^{**}2)/CPP(l) : l=1,..,N]. \qquad (4.1.1)$$

To make the L(j) comparable across multiprogramming sets, The L(j) calculated by equation 4.1.1 will be multiplied by the number of programs in the multiprogramming set, N. The page allocation weights can be used in this form since it is the relative sizes of the weights that is important.

Table 3. Page I/O Ratio Analysis

SOURCE	RAW REGR COEFFS	STD ERROR OF EST
GRAND MEAN	.603356	.085417
MPG SET	.329145	.241595
MPG SET	.617560	.241595
MPG SET	.239244	.241595
PRIORITY	-1.104013	.241595
PRIORITY	-1.083273	.241595
PRIORITY	-.917266	.241595
INTERACTION	-1.173274	.683333
INTERACTION	-1.567103	.683333
INTERACTION	-.423430	.683333
INTERACTION	-1.337678	.683333
INTERACTION	-1.306999	.683333
INTERACTION	-1.669633	.683333
INTERACTION	-.464727	.683333
INTERACTION	-1.358317	.683333
INTERACTION	-.564918	.683333

SOURCE	MEAN SQUARE	DF	F	P
GRAND MEAN	11.649	1	49.896	.0001
MPG EFFECT	0.521	3	2.231	.1241
PRIORITY	2.198	3	9.413	.0009
INTERACTIONS	0.263	9	1.128	.3985
ERROR TERM	0.233	16		

Both replications of Paging Estimation runs with 4, 5, 6, and 7 programs were used giving 44 observations of L(j). The data for these runs is given in the Appendix, Tables A9 and A10. The equalized estimates for L are given in Table 5 and the ANOVA table for a 4X4 factorial design is given in Table 6. Tests of hypothesis 10 use only the first four programs in the analysis.

Examination of the ANOVA table reveals a significant interaction at the .0001 level, hypothesis 10 is not supported. Regression coefficients from this analysis

Table 4. System-wide Paging Ratio Analysis

CELL	CELL MEANS	CELL STD DEVS
N = 4	0.257946	3.380890E-02
N = 5	0.477944	3.876854E-02
N = 6	0.563385	1.867964E-02
N = 7	0.354059	6.027378E-03

SOURCE	MEAN SQUARE	DF	F	P
LINEAR	0.014	1	18.436	.0128
QUADRIATIC	0.092	1	121.610	.0004
CUBIC	0.003	1	3.387	.1396
ERROR TERM	0.001	4		

cannot be used since priority 5, 6, and 7 programs have not been included. Regression analysis was therefore performed using all 44 observations with covariates: multiprogramming level (N), priority (j), priority squared (j**2), multiprogramming level squared (N**2), and the cross-product of multiprogramming level and priority. The coefficients resulting from this analysis are given in the bottom cf Table 6. These coefficients are used in computing the weights used in the predictive model.

Since there are significant deviations from unity in the ANOVA it can be concluded that the paging model is deficient. At this point it will be assumed that the working set weights that have been estimated are not specific to the programs used in the analysis but reflect error in the paging submodel and continue with the analysis.

Table 5. Estimated Working Set Weights

PRIORITY (j)	MULTIPROGRAMMING SET (N) 4	5	6	7
1	2.140348	0.699155	0.375096	0.355586
1	1.799340	0.862835	0.377862	0.368515
2	0.384188	0.323695	0.205488	0.162483
2	0.552756	0.363280	0.230628	0.175931
3	0.421240	0.418820	0.367578	0.279153
3	0.467356	0.360490	0.268770	0.273000
4	1.054224	0.708645	0.519882	0.411845
4	1.180548	0.675260	0.475926	0.426566
5	. . .	2.849685	1.347792	1.815296
5	. . .	3.098625	1.236636	2.007369
6	. . .	. . .	3.184158	1.917405
6	. . .	. . .	3.410190	1.985942
7	. . .	. . .	. . .	2.059225
7	. . .	. . .	. . .	1.762670

Miscellaneous Parameters

Prior to using the model to predict program performance, the CPU rates (MU), paging indices (A), maximum storage (S) and critical memory point (M) must be estimated. For this purpose the experimental programs were executed in 8 runs of N=4, with each experimental program being executed with 3 copies of the control program. The measured and calculated results of these runs are given in appendix A, Table A11 and A12.

Table 6. Working Set Weight Analysis

SOURCE	MEAN SQUARE	DF	F	P
GRAND MEAN	9.774	1	1499.218	.0001
MPG EFFECT	0.802	3	122.997	.0001
PRIORITY	0.590	3	90.430	.0001
INTERACTIONS	0.196	9	29.991	.0001
ERROR TERM	0.007	16		

SOURCE	RAW REGR COEFFS	STD ERROR OF EST
PRIORITY (j)	-0.286627	0.344902
j*j	.143958	0.035327
j*N	-0.060599	0.065588
MPG EFFECT (N)	.180209	1.019103
N*N	-0.025178	0.094479

CPU Rate Estimation

CPU rates for the control programs were estimated from the 4 runs of the control program with N = 4, 5, 6 and 7. The parameter MU was found to vary considerably, even within a single run. This effect is thought to be related to the method used to account for program time and the processing which takes place following an interrupt and before the value of the CPU clock is stored. This has the effect of making CPU service intervals for low pricrity programs appear larger and therefore the CPU rates (MU) would appear smaller.

The rates computed from each of the runs in appendix A, Table A9 and A10 are weighted averages of the rates computed from each program with I/O access rates used as weights. The expansion factors (CYT) were used to increase these rates (decrease service times) by applying the

equations in section 2.5, approximating the instantaneous I/O rates by LAM = 1/ET*(1-CP). The rates from the 4 runs were then averaged together and used in subsequent calculations and in the predictive model.

The CPU rates for the experimental programs were then estimated from the 8 runs in appendix A, Tables A11 and A12 The same procedures as before were used to estimate CYT and thereby remove some of the effects of "cycle stealing" due to I/O.

Virtual Storage Estimation

This turned out to be trivial since the programs were constructed to have the same maximum sizes (162K). This number was taken from the jcb listings which gives a measurement of the maximum virtual storage used.

Storage and Paging Index Estimation

These parameters were estimated in a sequential fashion. First a "quasi" paging index A' was computed with the ratio of the sum of critical storage to real storage equal to 1. We have

$$A' = w(j) * PI(j) / CP(j) * K * (S(j) - w(j)). \qquad (4.1.2)$$

A linear model was then fitted to this data using ln(A') as the dependent variable and ln(N) as the independent variable. Symbolically we have A = A'*(N*M/R)**-q or A =

A'*((M/R)**-q)*N**-q. The choice for M = 56 was justified on the basis of what follows. If the paging rate for the entire system is less than 2 pages a second it may be concluded that memory constraints are not active. Depending on the length of a program run, an apparent system paging rate of 2 pages/second can be caused by the initial flurry of activity when the programs begin execution and open files. The fact that the constraint is not active at N = 4 may be inferred from the size of the working sets of the low priority programs compared to the higher priority programs. The memory constraint appears to be just barely active at N = 5 so an appropriate value for M is M = R/5 = 280/5 = 56K.

Using M = 56K and equation 4.1.2, A is estimated for 4 different priorities of the CNTL program. From Table 7 it may be observed that the values of A vary from a low of 0.000504336 to a high of 0.21720546. This wide variation in the values of A are again indicative of the paging submodel's failure to accurately predict paging rates.

To determine what is a reasonable value for A, dimensional analysis is applied to the defining equation for paging.

The dimensions of interest are those of K and A. In the estimation process, a value of K = 490 was used for the S/370 model 148 (meaning that the 148's average speed is 490,000 instructions per second). This makes the units of K instructions per thousanths of a second and therefore the

Table 7. Paging Index Calculations

MPG LVL (j)	A' VALUES FOR M = 56K PRIORITY LEVELS 1	2	3	4
4	0.001343	0.000504	0.001292	0.008048
5	0.002593	0.002228	0.002144	0.004502
6	0.028177	0.029452	0.044289	0.070589
7	0.051738	0.045349	0.070277	0.119278
CORR	0.962743	0.977337	0.942172	0.849545
b	-16.743895	-19.677474	-18.106903	-13.270179
k	7.110351	8.672056	7.998743	5.655085
A	0.004301	0.002735	0.004414	0.013748
NORMALIZE	1.572647	1.000000	1.613795	5.026601

MPG SET (N)	NORMALIZATION FACTORS PRIORITIES (j) 1	2	3	4	5	6	7
4	2.663	1.000	2.562	15.957	. . .	. . .	. . .
5	1.164	1.000	0.962	2.021	2.345	. . .	. . .
6	0.957	1.000	1.504	2.397	3.591	7.375	. . .
7	1.141	1.000	1.550	2.630	4.707	4.556	4.425

units of A are in memory references per instruction divided by 1000. Expressing A in terms of memory references per instruction for the A-values from Table 7, A = 4.3, 2.7, 4.4 and 13.7 respectively.

System /370 instructions reference either 0, 1, 2 or 3 locations in main memory. Of these instruction, the ones that reference 0 storage locations are: supervisory call instructions, register instructions and those used for integer and floating point arithmetic. These are not frequently used in commercial programs and, even when they are used they must be used with instructions which reference 1 storage location to retrieve and store calculated data.

The instructions which reference 3 storage locations are also infrequently used. These are instructions which actually reference two starting addresses but involve movement of enough data that ocassionally the data overlaps multiple pages. An example of an instruction of this type is the long move instruction.

There are a large number of instructions which reference a single location and they are very frequently used. Instructions in this class include branching, register loading and storing and immediate instructions which contain a single byte of data in the instruction. Instructions which reference 2 storage locations are also very frequently used and these include storage to storage moves and compares. These instructions are particularly frequent in commericial applications where they are used extensively for logic and data formatting.

Based on the previous discussion and observaticns of instruction profiles of many programs it is very likely that the A value for most programs would be in the range of 1 to 2 storage references per instruction. The smallest cf the estimates for A in Table 7 (and the only one in the realm of possibility) indicates 2.73 storage references per instruction. Although this is possible it is not very likely. However the above analysis does point out that the only estimates for paging index in the practical range are those developed from partition 2 cf Table 7. In order to make A

values estimable using this format, it was decided to normalize A' estimates by the corresponding factors from the bottom of Table 7 by dividing by the appropriate factor. These factors were derived by dividing the A' calculated in each partition of the four runs of the CNTL program by the corresponding estimate for A' from partition 2 (column 2 in the table).

To estimate the page indices for the experimental programs, 8 runs of 4 programs each were performed. These runs consisted of a copy of the experimental program and 3 copies of the CNTL program. The measured and calculated results of these runs (see appendix A Tables A11 and A12) and the conversions in Table 7 were applied to compute the estimates of A in Table 8.

From the page index estimates of Table 8 it may be observed that the paging manipulation was not successful since the estimates for A are apparently correlated with low I/O variance rather than with the high paging indicators (1XX in the program identification). From the program estimation runs in the appendix, Table A10 it may also be observed that higher CPU service times are apparently correlated with high priority execution. Along with this effect a slight increase in process time may be observed, however the increase in process time is not proportional to the increase in the number of I/O operations, yielding a higher CPU rate. Fewer I/O's are reported at higher priori-

ties because the lower priority jobs were started first in order to delay some of the effects of priority on job initiation.

At the same time the lower priority jobs are delayed in starting to execute so it makes no sense to use observations during this initiation stage. Thus only those measurements which are taken while all programs are executing are included. This procedure should yield results which are contrary to the "expansion effect" and also contrary to the argument that the lower priority programs should have smaller rates (large average service times) because they include "start-up" time and time for initially opening the files. While this effect appears to be systematic the exact cause is unknown at this time.

Experimental Manipulations

To the extent that the paging manipulation does result in higher paging rates the manipulation was successful.

However the previous discussion on page index estimation points out that the attempt to manipulate page index was not successful in terms of the estimates of A since this index is correlated with the I/O variance manipulation and not with the paging manipulation.

Table 8. Experimental Program Estimates

PROGRAM	N	j	MU	A	DISK	FILES	
000	6	1	126.584	0.002069	3	1	4
011	6	1	126.800	0.016921	3	2	3
101	6	1	125.073	0.020515	3	2	3
110	6	1	125.528	0.003124	3	1	4
001	6	4	128.534	0.002655	3	2	3
010	6	4	128.350	0.000941	3	1	4
100	6	4	128.764	0.001193	3	1	4
111	6	4	127.095	0.002889	3	2	3
001	7	1	128.534	0.002655	3	2	3
010	7	1	128.350	0.000941	3	1	4
100	7	1	128.764	0.001193	3	1	4
111	7	1	127.095	0.002889	3	2	3
000	7	4	126.584	0.002069	3	1	4
011	7	4	126.800	0.016921	3	2	3
101	7	4	125.073	0.020515	3	2	3
110	7	4	125.528	0.003124	3	1	4
CNTL	6	2	54.296	0.002735	2	3	4
CNTL	6	3	54.296	0.002735	2	5	6
CNTL	6	4	54.296	0.002735	2	7	8
CNTL	6	5	54.296	0.002735	4	1	2
CNTL	6	6	54.296	0.002735	4	3	4
CNTL	6	1	54.296	0.002735	2	1	2
CNTL	6	2	54.296	0.002735	2	3	4
CNTL	6	3	54.296	0.002735	2	5	6
CNTL	6	5	54.296	0.002735	4	1	2
CNTL	6	6	54.296	0.002735	4	3	4
CNTL	7	2	54.296	0.002735	2	3	4
CNTL	7	3	54.296	0.002735	2	5	6
CNTL	7	4	54.296	0.002735	2	7	8
CNTL	7	5	54.296	0.002735	4	1	2
CNTL	7	6	54.296	0.002735	4	3	4
CNTL	7	7	54.296	0.002735	4	5	6
CNTL	7	1	54.296	0.002735	2	1	2
CNTL	7	2	54.296	0.002735	2	3	4
CNTL	7	3	54.296	0.002735	2	5	6
CNTL	7	5	54.296	0.002735	4	1	2
CNTL	7	6	54.296	0.002735	4	3	4
CNTL	7	7	54.296	0.002735	4	5	6

Note: Program definitions are given in Figure 9.

CPU Variance Manipulation

From the closeness of all estimates for CPU service time (rate) it may be inferred that the manipulation for CPU service variance was successful although the measurement tools used in this research do not permit estimation of the absolute magnitude. The following formulas were used to construct the CPU variance manipulations:

```
S=(1+X/K)/MU**2,
V(S)=v*(v+1)/3*(K*MU)**2,                    (4.2.1)
E(S)=1/MU,
```

where K is an integer representing the number of times the compute "kernal" is to be executed and X is an integer in [-v,v] chosen with probability 1/(2*v+1), and v is an integer less than or equal to K. In all cases K = 2 is used. For the low variance programs (X1X), v = 0 is used for zero variance. For the high variance programs, (X0X) v = 2 was used for a variance of 1/2*MU or approximately 0.000032746 seconds squared compared to a mean service time of 0.00784314 seconds. For the CNTL program v = 1 was chosen. This yields a variance of 1/6*MU or 0.00005653 seconds squared compared to a mean service time of approximately 0.0184176 seconds.

I/O Variance Manipulation

The I/O variance was again manipulated by construction. The high I/O variance program files were set up with their centers 15 cylinders apart and file sizes of 9 cylinders. The low I/O variance programs were set up with their centers 3 cylinders apart and with file sizes of 3 cylinders each. The I/O variance for the low and high variance programs are approximately 93 Ms squared and 123 Ms squared respectively. However, the means are alsc different in this case, approximately 28 Ms for the high variance programs and 18 Ms for the low variance programs.

Model Predictions

Sixteen runs of the APL model were made to develop the predictions to be used in the analysis of variance. The results of these runs for the experimental programs is given in Table 9. In making these predictions the APL model was found to converge quite rapidly. Of the 16 runs, 13 converged in 6 iterations, 2 converged in 10 iteraticns and one required 16 iterations.

The convergence criteria was to stop iterating when the maximum absolute difference in I/O rates from one iteration to the next was less than 0.002. This results in an difference of less than 0.002 seconds in I/O service time for the very lowest priority programs. A diagram of the relationship between the various submodels and the

Table 9. Model Predictions

PROG	M L	P R	CPU UTIL	CYCLE TIME	PAGE RATE	PAGEIN RATE	I/O RATE	CHNL UTIL	OVER HEAD	WRKG SET
000	6	1	0.21	0.034	3.5	2.9	26.65	0.033	0.23	53
000	7	4	.04	.114	5.6	4.1	4.60	0.004	.34	24
001	6	4	.05	.131	2.3	1.7	5.79	0.007	.21	30
001	7	1	.18	.032	9.6	8.3	22.90	0.018	.36	57
010	6	4	.05	.138	1.5	1.1	6.08	0.008	.20	19
010	7	1	.19	.034	6.5	5.5	23.47	0.019	.33	36
011	6	1	.22	.031	5.8	4.8	26.85	0.034	.26	111
011	7	4	.02	.097	10.4	7.6	2.65	0.002	.37	46
100	6	4	.05	.137	1.7	1.2	5.95	0.007	.20	22
100	7	1	.18	.034	7.3	6.3	22.65	0.018	.34	40
101	6	1	.22	.031	5.7	4.7	26.76	0.034	.27	119
101	7	4	.02	.096	10.9	8.0	2.40	0.002	.38	48
110	6	1	.21	.034	4.0	3.3	26.18	0.033	.24	60
110	7	4	.04	.111	6.3	4.6	4.30	0.003	.35	27
111	6	4	.05	.130	2.5	1.9	5.75	0.007	.21	32
111	7	1	0.18	0.032	10.1	8.7	22.50	0.017	0.36	60

Note: Program definitions are given in Figure 9.

convergence algorithm is given in Figure 10.

Model Validation

The experimental programs were then run on the System /370 Model 148. Since each run was repeated there were a total of 32 runs. The measured and the calculated performance variables from the experimental programs are given in Tables 10 and 11.

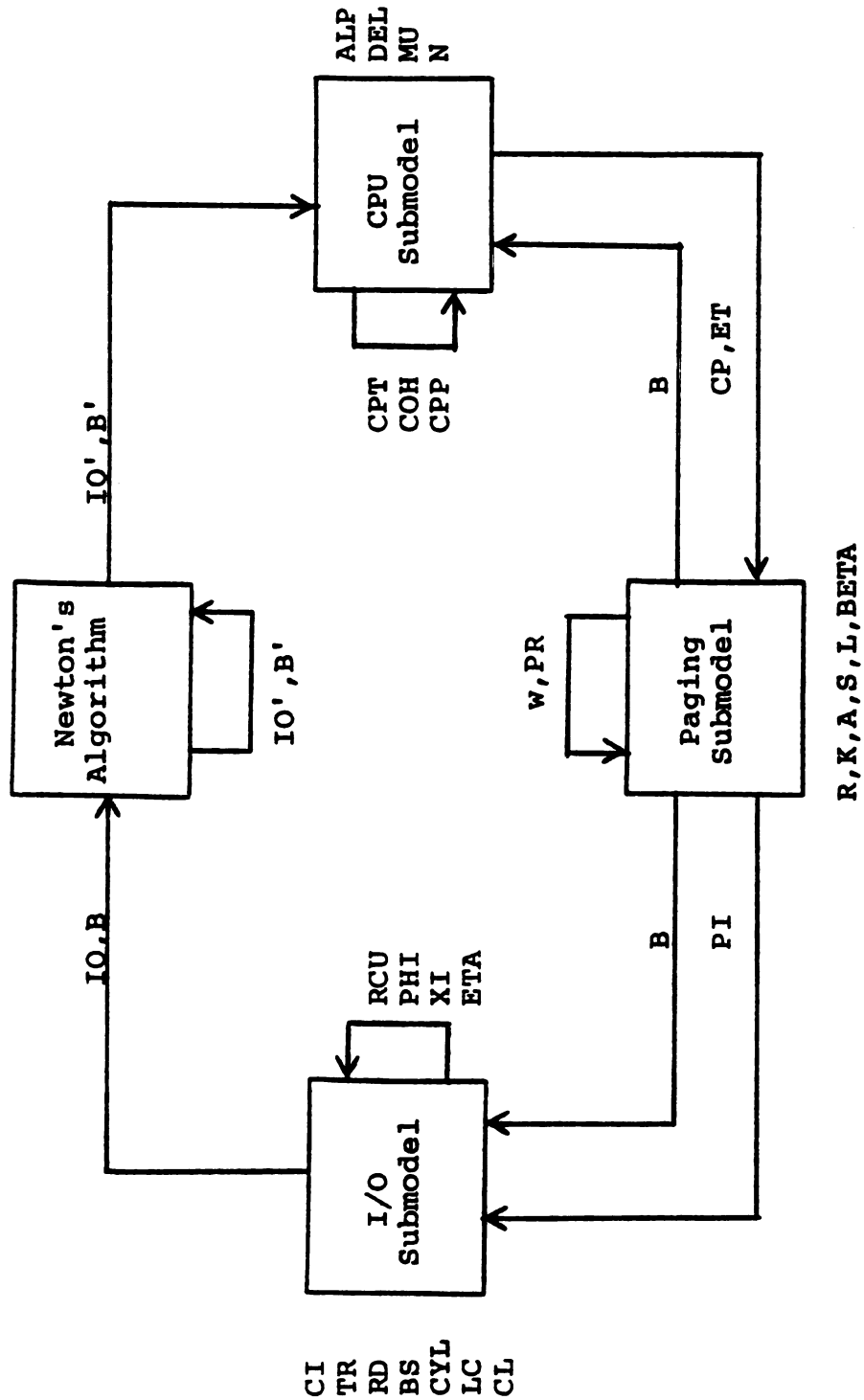

Figure 10. Multiprogramming Interaction Model

Table 10. Experimental Runs - Measured

PROG	M L	P R	PAGES OUT	PAGES IN	OTHER I/O	ELAPSED TIME	PROC TIME	OVER- HEAD	WRKG SET
000	6	1	99	303	8,720	342.09	69.75	69.62	50
000	6	1	68	203	8,720	330.70	69.56	65.29	52
000	7	4	1,241	7,413	8,720	1,859.87	72.39	442.99	42
000	7	4	821	5,409	8,462	1,511.70	69.30	362.51	42
001	6	4	234	1,277	8,718	833.58	69.23	176.30	52
001	6	4	288	1,423	8,718	842.08	69.31	182.45	50
001	7	1	128	1,009	5,800	322.98	47.06	75.22	54
001	7	1	217	1,462	8,353	469.79	67.93	110.99	48
010	6	4	197	1,172	8,720	848.99	69.32	182.52	54
010	6	4	182	1,048	8,720	880.52	69.18	181.84	46
010	7	1	125	1,574	7,709	489.00	63.11	115.97	48
010	7	1	106	1,479	8,057	478.64	65.76	114.60	54
011	6	1	119	426	8,718	330.65	70.36	65.86	48
011	6	1	142	419	8,718	325.86	70.55	67.47	52
011	7	4	815	5,086	7,331	1,365.27	60.97	325.92	46
011	7	4	950	5,700	6,765	1,455.22	56.81	344.14	44
100	6	4	919	4,413	8,720	1,266.70	70.30	296.02	58
100	6	4	1,191	5,808	8,720	1,463.46	70.93	347.36	56
100	7	1	223	3,537	8,498	743.75	69.70	178.52	54
100	7	1	275	4,008	8,513	819.89	70.10	196.13	62
101	6	1	127	578	8,718	337.04	70.57	70.14	58
101	6	1	129	525	8,718	330.45	70.49	69.06	60
101	7	4	1,862	7,220	1,329	1,460.52	13.25	346.28	38
101	7	4	1,641	6,061	1,060	1,244.75	10.70	292.30	36
110	6	1	83	492	8,720	367.05	70.33	78.89	58
110	6	1	84	409	8,720	358.19	70.47	74.85	58
110	7	4	1,693	6,133	927	1,257.55	9.47	295.34	40
110	7	4	1,627	6,206	1,224	1,254.70	11.85	296.95	42
111	6	4	708	3,477	8,718	1,103.52	71.18	255.99	56
111	6	4	678	3,482	8,718	1,100.18	70.92	254.93	56
111	7	1	241	2,618	8,601	620.79	71.15	147.87	56
111	7	1	259	2,468	8,432	593.80	69.73	141.51	54

Note: Program definitions are given in Figure 9.

Table 11. Experimental Runs - Calculated

PROG	M L	P R	CPU UTIL	CYCLE TIME	PAGE RATE	PAGEIN RATE	I/O RATE	CHNL UTIL	OVER HEAD	WRKG SET
000	6	1	0.20	0.038	1.2	0.9	25.49	0.021	0.20	50
000	6	1	.20	.037	0.8	0.6	26.37	0.021	.20	52
000	7	4	.04	.115	4.7	4.0	4.69	0.015	.24	42
000	7	4	.05	.109	4.1	3.6	5.60	0.014	.24	42
001	6	4	.08	.083	1.8	1.5	10.46	0.012	.21	52
001	6	4	.08	.083	2.0	1.7	10.35	0.012	.22	50
001	7	1	.15	.047	3.5	3.1	17.96	0.022	.23	54
001	7	1	.15	.048	3.6	3.1	17.78	0.022	.24	48
010	6	4	.08	.086	1.6	1.4	10.27	0.011	.22	54
010	6	4	.08	.090	1.4	1.2	9.90	0.011	.21	46
010	7	1	.13	.053	3.5	3.2	15.77	0.020	.24	48
010	7	1	.14	.050	3.3	3.1	16.84	0.020	.24	54
011	6	1	.21	.036	1.6	1.3	26.37	0.023	.20	48
011	6	1	.22	.036	1.7	1.3	26.76	0.023	.21	52
011	7	4	.05	.110	4.3	3.7	5.37	0.015	.24	46
011	7	4	.04	.117	4.6	3.9	4.65	0.015	.24	44
100	6	4	.06	.096	4.2	3.5	6.89	0.016	.23	58
100	6	4	.05	.101	4.8	4.0	6.00	0.016	.24	56
100	7	1	.09	.062	5.1	4.8	11.43	0.020	.24	54
100	7	1	.09	.065	5.2	4.9	10.38	0.019	.24	62
101	6	1	.21	.036	2.1	1.7	25.87	0.024	.21	58
101	6	1	.21	.036	2.0	1.6	26.39	0.024	.21	60
101	7	4	.01	.171	6.2	4.9	0.91	0.016	.24	48
101	7	4	.01	.175	6.0	4.9	0.85	0.016	.24	48
110	6	1	.19	.040	1.6	1.3	23.76	0.021	.22	60
110	6	1	.20	.039	1.4	1.1	24.35	0.021	.21	60
110	7	4	.01	.178	6.2	4.9	0.74	0.016	.24	27
110	7	4	.01	.169	6.2	4.9	0.98	0.017	.24	27
111	6	4	.07	.090	3.8	3.2	7.90	0.015	.23	32
111	6	4	.06	.090	3.8	3.2	7.93	0.015	.23	32
111	7	1	.12	.055	4.6	4.2	13.86	0.021	.24	60
111	7	1	0.12	0.054	4.6	4.2	14.20	0.022	0.24	60

Note: Program definitions are given in Figure 9.

The predictions and these results were then merged and made inputs to an analysis of variance using Finn's MULTIVARIANCE [22]. The criterion variables were transformed into a relative error by dividing the differences between the predictions and the outcomes by the outcomes.

Results of Analysis of Variance

Hypotheses 1-6. Examination of the Multivariate Analysis of Variance results (Table 12) indicates significance for all 10 interactions. This prevents tests for main effects because the main effects are confounded with the interactions. Based on the analysis of variance it could well be that there is not a fixed effect but this cannot be shown using these tests. This effectly bars statements about the fixed effects of the five factors other than to say that there is apparently significant errors in the predictive model. Thus these techniques are unable to help further in analyzing the model. Univariate techniques will now be explored for some partial answers.

Table 12. ANOVA Table of Prediction Errors

CP	ET	PI	PO	RCU	COH	W
LEAST SQUARES ESTIMATES OF EFFECTS-GRAND MEAN						
.3236	-.0636	.9023	.8494	-.2218	.3149	-.0369
STD ERRORS OF LEAST SQUARE ESTIMATES - GRAND MEAN						
.0181	.0044	.0436	.0486	.0020	.0372	.0091

STEP-WISE REGRESSION ANALYSIS

GRAND MEAN	F(7,10) = 2,268.646	P<.0001
PAGE INDEX	F(7,10) = 154.835	P<.0001
CPU VARIANCE	F(7,10) = 4.626	P<.0150
I/O VARIANCE	F(7,10) = 128.399	P<.0001
MPG LEVEL	F(7,10) = 3,296.047	P<.0001
PRIORITY	F(7,10) = 6,596.086	P<.0001
PGNDX X CPVAR	F(7,10) = 32.045	P<.0001
PGNDX X IOVAR	F(7,10) = 17.399	P<.0001
PGNDX X MPG	F(7,10) = 41.494	P<.0001
PGNDX X PRIORITY	F(7,10) = 56.054	P<.0001
CPVAR X IOVAR	F(7,10) = 23.016	P<.0001
CPVAR X MPG	F(7,10) = 27.486	P<.0001
CPVAR X PRIORITY	F(7,10) = 6.103	P<.0057
IOVAR X MPG	F(7,10) = 32.918	P<.0001
IOVAR X PRIORITY	F(7,10) = 36.309	P<.0001
MPG X PRIORITY	F(7,10) = 376.800	P<.0001

Confidence Interval Analysis

Hypotheses 7-13. To help assess the effectiveness of the model, confidence interval analysis will now be employed. The null hypotheses will be supported if the confidence intervals calculated from estimates of the of the grand mean of the relative error lies in the interval [-.15,+.15]. The confidence intervals are of the form u±t*s/n**.5, where u is the estimate of the mean, and t is the value with 16 degrees of freedom at the .025 probability level (2.12), n is the number of observations (32), and s is the standard error of the estimate [22]. The confidence intervals for the dependent variables are as follows:

PROGRAM CPU UTILIZATION (CP):	[0.282928,0.364252]
PROGRAM CYCLE TIME (ET):	[-.073485,-.053715]
TOTAL PAGING RATE (PR):	[0.740463,0.958317]
PAGING INPUT RATE (PI):	[0.804467,1.000070]
CHANNEL UTILIZATION (RCU):	[-.226226,-.217274]
CPU OVERHEAD (CHT):	[0.231618,0.398242]
WORKING SET (w):	[-.057384,0.016416]

From this it can observed that hypotheses 8 and 13 are supported but hypotheses 7, 9, 10, 11, and 12 do not receive support. Thus from a statistical point of view, the model does not predict the performance variables of interest within the desired error margin of 15%.

Sources of Error

CPU Submodel

A possible source of error might have been the fact that the CPU submodel does not explicitly account for variance in I/O service time. The I/O service time depends on a random pattern of accesses by device, channel contention, rotational position sensing, and even device errors and retry. Minor sources of variation in CPU service times may be attributable to error routines and exception routines.

A potential source of error in the CPU submodel is the fact that the model does not address the portion of CPU overhead which is not interruptible, nor does it handle the portion of I/O processing which can interrupt higher priority programs.

A major source of error is the scaling effect due to differences in small values of CPU utilization. For example, the difference between a predicted CPU utilization of 0.02 and an observed utilization of 0.01 is of little practical importance but the relative error of 100% seriously biases the CPU utilization statistic.

Potential Sources of Error-Paging

The major source of error in the model is the subsystem which deals with paging. Most previous authors have either considered the paging rate to be fixed or ignored it altogether. Some authors have assumed the rate of paging operations to be proportional to the instruction execution rate for the task itself (this approach ignores data references). Others predict paging rates but only for certain "safe" values (where the paging "function" is more or less linear).

The choice of two factors, critical storage size and page index to represent a program's paging behavior is an oversimplification. A program's address space may consist of many distinct types of data, each of which may have a different reference pattern. For example, the executable instructions of a program may be composed of loops, jumps, or straight-line code and the instructions may or may not modify themselves. The data portion may be scanned sequentially or searched using a binary or indexed technique. Data areas may be separate from instructions or the data may be interspersed among the instructions, a situation referred to as locality of reference.

The use of a program's full size as a parameter of page demand is somewhat unrealistic in view of the fact that a certain portion of many programs are composed of error routines, initialization and termination routines which

execute an insignificant amount of time.

The proposed model includes some of the methods of previous authors, and therefore includes some of the shortcomings. In particular the paging coefficients estimated for the page-in and page-out operations may be more representative of the characteristics of the programs run on the system than they are of the operating system and memory size of the system. If this is indeed the case, the error in predicting the performance of a particular set of programs will vary according to their deviation from the "typical" programs in an installation.

The assumption that CPU overhead processing is carried out in non-pageable memory is contrary to the known operation of the operating system used in these experiments. Some portions of the operating system used in this research compete with the user programs for the available real storage pages.

A final source of error in the paging submodel is that the paging indices were estimated at a multiprogramming level of N=4. From previous comments about the lack of a real memory constraint at N=4, it is likely that more stable estimates for this parameter could be obtained at higher multiprogramming levels.

Post-Experimental Paging Analysis

Revised Paging Submodel

The results of the program runs with 4, 5, 6, and 7 copies of the control program were fitted to a model of the form w(j) = Q*CP(j)**1/k. This model was found to account for 6%, 69%, 99%, and 97% of the variation in w(j) for N = 4, 5, 6, and 7 respectively. The constant of proportionality (Q) was found to fit the expression R/sum[CP(l):l=1,..,N] very closely. This suggests a general expression for working set size of of the form

$$w(j)=Q*(A(j)*CP(j))**1/k, \qquad (4.5.1)$$

where Q = R/sum[(A(l)*CP(l))**1/k:l=1,..,N]. Working backward from this expression, the following expressions are suggested for the real-time paging rate:

$$PI(j)=CP(j)*A(j)*K*EXP(N)/(k-1)*w(j)**(k-1), \qquad (4.5.2)$$

$$EXP(N)=(sum[M(l):l=1,..,N]/R)**q.$$

The value of q was estimated to be 7.14 and the value of k was estimated to be approximately 4. Subsequent runs of the predictive model with the revised paging submodel and new estimates of A for N=7 yielded the following mean errors:

CP	ET	PR	PI	RCU	OHT	w
0.285	0.006	0.343	0.298	0.179	0.068	0.010

This significantly reduces the errors in paging and the size of the relative errors in CPU utilization (CP) and channel utilization (RCU) is believed to be due to the scaling effect mentioned earlier.

Comparison with Earlier Models

Fitting the control program runs to the Belady-Kuehner "lifetime function" [6] accounts for 47%, 47%, 98%, and 96% of the variation in paging rates. Likewise the paging expansion factor EXP(N) was found to account for 96% of the variation in the system paging rate. The large exponent value (7.14) was unexpected in view of quadratic and linear segments in Bard's Paging Index [3]. Chamberlain, Fuller, and Liu's Half-life function [12] was found to account for 49%, 39%, 99%, and 96% of the variation in paging rates when fitted to the data from the control program runs.

V. CONCLUSIONS AND RECOMMENDATIONS

Research Conclusions

The knowledge gained by study of the model discussed herein is a step toward better predictive methods for computer performance, I/O configuration performance and program performance.

The system overhead due to handling I/O requests was found to be proportional to the I/O rates and there was a small but significant positive effect due to program priority. There was a small but significant negative effect due to multiprogramming level. The paging overhead was fcund be proportional to paging rates but the small effects due to priority and multiprogramming levels were not statistically significant.

The increase in the ratio of page-writes to page-reads was not significant but there was a significant increase with increasing priority.

The method of sucessive estimation and the convergence techniques used in this model were found to be very effective. Convergence was usually obtained in less than 6 iterations. Improvements in the algorithm made after the

experiments had been performed reduced the number of iterations to the range of 4 to 5 iterations for about half of the model executions.

While the original goals of this research have not been achieved, positive benefits have been gained from this effort. For one thing the least elaborate part of the entire model, the paging submodel, has turned out to be the source of much of the error. Part of the reason why this part of the model is so simple is because this area is not as well understood as the operation of the CPU and I/O devices. Failure of the model to accurately represent paging behavior has a more serious effect on some of the criterion variables than others. For example, the criterion variables that seem to have the largest errors are paging counts (as expected) but also CPU overhead. This is not so hard to understand in view of the demonstrated effect of paging rates on CPU overhead.

The techniques employed with the variable paging manipulation of the synthetic programs can be used as a tool in controlling loads on other systems during performance measurement, systems tuning and benchmarks. The CPU and I/O submodels are well elaborated and it is possible that they can be of benefit to others who will study the kinds of systems studied in this research. The reasons why these models are so elaborate is that suitable models for this research were not generally available and the insights

gained from the model building process stimulated further elaboration.

Failure to accurately predict paging behavior does not prevent the model in this thesis from explaining many other events such as CPU and I/O service intervals, initial wait and completion times since paging rates are a matter of measurement in existing systems. It is possible that the use of the revised paging submodel can make this methcd worthwhile for practical applications. The inherent errors in this submodel at unconstrained memory levels is not too serious a defect since working set estimates are usually unimportant at unconstrained memory levels and the paging rates in these cases is usually neglible. Another benefit of this effort is that it clearly points out the direction that further research in this area ought to take.

Recommendations for Further Research

Clearly the direction for future research in this area is the development of better analytical models of paging behavior. A likely candidate for this research would be some variation of Belady and Kuehner's lifetime function approximation [6]. Another possibility is to validate the analytic paging model based on the relationship between the CPU utilization and working set size. Pcst-experimental analysis of the data obtained in this research revealed strong evidence for a relationship of the form w=Q*CP**1/k.

This could become the basis of more research into paging in priority interrupt driven systems.

The page-write/page-read ratios were not central to this research, however questions posed here suggests further study in this area. There is no reason why the data for such a study cannot be drawn from the job accounting data collected from production systems rather than from experiments.

Methods of analysis other than factorial analysis of variance is recommended since there is some indication that interactions will reduce the usefulness of this kind of analysis. One way to overcome this problem is to perform enough observations so that independent one-way analyses can be performed. A larger number of sample points can be obtained with fewer actual program runs by including <u>all</u> programs from each run in the analysis.

The size of the relative errors in predicting the cycle time (ET) and the system overhead (OHT) suggests that the large relative errors in the program CPU utilization (CP) and the channel utilization (RCU) are scaling effects. It is therefore recommended that these variables should be scaled so that the contribution to overall error of differences in small values is proportional to their importance.

It is also recommended that smaller compute kernals be used for synthetic programs. This would allow a greater

range and control of the paging rates and the CPU variance. For example compute kernals of about 20% the size of the ones used in this research would have been more apprcpriate and would have allcwed a greater range of CPU variance.

The needs expressed in the beginning of this thesis will continue to inspire the investigation necessary to the accomplishment of better ways to perform capacity management, resource scheduling, and data processing system tuning.

GLOSSARY OF TERMS

access arm: The mechanism which is used to physically position the read/write heads of disk-type devices for I/O operations.

application program: A program which serves an end-use function in a computing system rather than a utility function. Examples are; payroll, inventory, etc.

benchmark: Execution of a prescribed set of programs on one or more different computer systems for the purpose of performance measurement or comparison.

biased page allocation: A method of partitioning the pageable real memory of a virtual computer system which arbitrarily favors certain programs over others with respect to minimization of the system paging rate.

buffering: A method of I/O data management which reads or writes data from two or more areas of real memory, allowing CPU and I/O operations of the same program to be overlapped.

central server: The node of a queuing network which represents the CPU in some program models. The other nodes of the network represent the I/O devices.

channel: The data path between an I/O device or control unit and the main memory of a computer system.

completion time: The average total elapsed time from the instant a program accesses the CPU until the program generates an I/O or paging operation.

compute kernal: A collection of machine (computer) instructions which is artificially constructed to represent a "typical" program or to serve as a basis for comparisons in benchmarks.

CPU bound: A situation which prevails when the CPU is the limiting resource in a computer system. In this thesis a program is considered to be CPU bound if its CPU utilization exceeds 50% when no other programs are active on the system.

configuration: A particular combination of memory, processors, I/O devices and channels in a computer system.

cylinder: The collection of all records which can be read or written with one physical positioning of a disk access mechanism (arm).

direct access device: An I/O device with the ability to read or write data at any location on the recording medium without regard to sequence. Examples of direct access devices are disks and drums.

direct access method: The method of data access to disks or drums which takes advantage of the direct access characteristics of the device. This usually implies that the data is organized for direct access and the physical address of the data is generated by the program based on a randomizing algorithm.

disk scheduling: The queue management and service discipline for disk I/O requests. Examples of disk scheduling strategies are First-Come-First-Served (FCFS), Priority, and SCAN.

disk: A type of direct access device which uses iron oxide coated platters or "disks" as a recording medium. Data is recorded in concentric circles about the axis of a shaft which rotates the surfaces under read/write heads. A number of disk surfaces may be mounted on a single shaft.

dispatching: The process of storing the status (or state) of the program which has control of the CPU and restoring the status of a suspended program. The process of switching control of the CPU from one program to another.

EXCP: Execute-Channel-Program. A request by an executing program to the Operating System to commence a specific I/O operation.

FCFS: A queue service policy, First-Come-First-Served (see FIFO).

FIFO: A queue service policy, First-In-First-Out (see FCFS).

headway: The mean program execution interval between page faults.

I/O bound: A situation which prevails when the I/O devices are the limiting resources in a computer system. A program is considered to be I/O bound if it spends greater than 75% of its elapsed execution time waiting for I/O when it is the only program active on a computer system.

indexed access: A method of disk access which requires that the location of a particular disk record be retrieved from an index before the data record itself can be read. The index is a disk file but it may be made core resident during program execution.

initial wait: The elapsed time from the completion of a program's I/O until the program has access to the CPU. This implies that another program is using the CPU when the program in question becomes ready to run.

interrupt: An event which triggers a change in the program which is in control of the CPU. An interrupt may be voluntary, as in a request for I/O, or it may be involuntary, as is the case for a page exception or the completion of another program's I/O. In the latter case control will be ultimately given to the highest priority program that is ready to run.

IS: A queue service discipline for a server with infinite capacity, or an Infinite Server. This differs from the PS server in that the arrival of new requests does not diminish the rate at which previous arrivals are served.

job accounting: The process of system software collecting and recording I/O counts, paging counts, storage allocations and elapsed time and cumulative CPU time for the processes or programs in a computer system.

LCFS: Last-Come-First-Served queue service discipline. The most recently arrived request is served immediately and the earliest arriving requests that are still unsatisfied are not served until all later arrivals are served.

lifetime function: A relation between the amount of real memory allocated to a process (program) in a virtual system and the process time between page exceptions.

LRU: A paging algorithm which attempts to select the Least-Recently-Used page frame to satisfy a paging exception. A stack procedure which is used to implement the LRU algorithm.

method of false position: A method for accelerating convergence of a sequence by approximating the derivative in Newton's Method with a divided difference of previously computed sequence values.

module: The collection of disk surfaces which rotate about a common shaft in a disk I/O device. It is also referred to as a spindle.

multiprogramming: The prccess of concurrent execution of two or more programs in a single computer system.

multiprogramming set: The collection of all the programs or tasks that are concurrently executing in a single computer system during the interval under consideration.

non-pageable memory: The portion of high speed memory which is fixed and not allowed to be paged in a virtual system. Generally this portion of memory is used by the operating system for critical system tasks such as I/O or paging management.

occupancy: The probability of executing the instructions in a particular page of a program's address space.

overhead: The portion of elapsed time during which the operating system is performing functions on behalf of problem (user) programs. This time is not attributed to any specific program by job accounting.

page exception: An interrupt which occurs whenever reference is made to a page of virtual memory which is not in real memory, ie. page fault.

page fault: A page exception.

page: The smallest unit of virtual memory that is managed by the paging mechanism of the virtual system. In this thesis a page is fixed and either 2048 or 4096 bytes.

paging: The process of selecting a page frame of real memory, possibly writing its contents onto auxiliary storage (disk or drum), and reading the missing virtual page into the selected page frame in virtual systems.

peripheral: An I/O unit or piece of data processing equipment which is locally attached to the central processor (CPU) but is not a part of it.

priority scheduling: The operating system policy of giving control to the program which has the greatest importance (possibly cn an arbitrary scale) and is ready to run.

priority-resume service: The queue service discipline which services the highest priority arrival in the queue and resumes service to any interrupted arrival only after all higher priority arrivals have completed service.

problem state: The state of the CPU when user program instructions are being executed. In this state, certain CPU instructicns which may compromise the integrity of the entire system are prohibited. This is in contradiction to system state, during which any instruction may be executed.

product-form: The form of the solution for network queuing problems which expresses the probabilities for network and subnetwork states as the product of the probibilities for the individual nodes of the network. Such a problem is said to be separable.

PS: A queue service discipline for a server which has Processor Sharing. In this case the server can service all arrivals simultaneously, but with the individual service rates identical and inversely proportional to the number of arrivals being served. This construct is the limiting case for round-robin scheduling.

random access: See direct access method.

response: In the case of terminals, usually the elapsed time from data input until the answer or acknowledgement is received at the terminal. In the case of program I/O, response time is the time from the request to begin an I/O operation until the operation is completed.

RPS: Rotational Position Sensing. The technology which allows some disk or drum I/O units to disconnect from the channel during rotational delay and to reccnnect to the channel when the desired records come under the read/write heads. This greatly increases the data carrying capacity of the channel.

SATF: An I/O scheduling method for disks and drums which services a queue of I/O requests so that requests with the shortest access time (from the current angular displacement of the recording medium or the current sector) are serviced first,
Shortest-Access-Time-First.

SCAN: An I/O scheduling method for disks which moves the access mechanism back and forth over the disk surfaces, stopping at cylinders which have outstanding requests to be serviced.

scheduling: In this thesis, scheduling is understood to mean the arrangement and sequencing of programs or jobs for production as performed by a person or a program which is not a part of the operating system. In other uses of the word "scheduling", the meaning is made clear by the presence of another word such as "disk", "I/O", etc.

seek: A physical movement of a disk access mechanism or arm from one position or cylinder location to another.

Semi-Markoff process: A stochastic process characterized by the fact that the probability distribution of the time between changes in system states depends only on the initial and terminal states and is otherwise independent of the previous history of the system.

separable: A problem which can be formulated in such a way that the solution is expressible as a product of probabilities (see product form).

shared memory: Portions of virtual memory having subroutines which are executed by two or more independent programs. Such subroutines may not modify instructions or data areas in the shared memory, and are said to be re-entrant.

spindle: See module or disk.

spooling: A method of increasing overall computer system efficiency by freeing programs from direct interaction with low speed I/O devices. The input is read from data input devices by a "reader" program, blocked and written to disk. When the actual processing program is executed, its card and print output is blocked and written to disk. Actual punching and printing are accomplished asynchronously by "writer" programs.

SSTF: A disk scheduling method which services I/O requests in the order of the shortest seek times from the current location of the access arm, Shortest-Seek-Time-First.

system state: The state of the CPU when operating system instructions are being executed. In this state, all instructions are allowed including special system (privileged) instructions. Most of the time in this state is not attributed to any specific program and may be thought of as overhead.

time-sharing: The process of giving computer access to terminal users so that the user can interact with the computer with little regard to the absence or presence of other users. Time-sharing systems generally use the round-robin or time-slicing scheduling discipline. The service discipline rotates access to the CPU among the members of the queue in discrete increments of time.

transaction: A logical exchange of data between a customer or user and the data processing system. Examples of transactions are payments, orders, requests, requisitions and inquiries.

virtual system: A computer system (including hardware and software) which is capable of executing programs of any size without regard to the size of the computer's real memory.

working set: The collection of all pages of a program (or system) which has been referenced during some arbitrary interval.

APPENDIX A

APPENDIX A

Table A1. Non-paging Runs Repl I - Measured

PROG	P R	PAGES OUT	PAGES IN	OTHER I/O	ELAPSED TIME	PROCESS TIME	WRKG SET
01101	1	. . .	. . .	11,961	301.01	97.87	58
OVRHD					301.01	34.07	1136
01101	1	. . .	. . .	12,002	342.50	96.98	60
02011	2	. . .	. . .	6,334	342.50	117.44	56
OVRHD					342.50	53.16	1136
01101	1	. . .	. . .	11,900	357.18	95.62	60
02011	2	. . .	. . .	5,571	357.18	102.71	58
03001	3	. . .	. . .	2,811	355.12	52.45	54
OVRHD					355.12	59.24	1136
01101	1	. . .	. . .	11,712	431.66	93.67	62
02011	2	. . .	. . .	6,718	431.66	123.99	56
03001	3	. . .	. . .	3,736	431.66	68.68	56
04000	4	. . .	. . .	2,151	429.59	40.35	56
OVRHD					429.59	73.81	1136
01101	1	. . .	. . .	11,662	437.97	92.54	60
02011	2	. . .	. . .	6,507	437.97	119.86	58
03001	3	. . .	. . .	3,820	437.97	70.04	58
04000	4	. . .	. . .	2,332	437.97	43.94	58
05110	5	. . .	. . .	985	431.77	18.97	56
OVRHD					431.77	76.10	1136
01101	1	. . .	. . .	11,448	452.48	90.86	62
02011	2	. . .	. . .	6,673	452.48	122.83	58
03001	3	. . .	. . .	3,942	452.48	72.24	58
04000	4	. . .	. . .	2,469	452.48	46.40	60
05110	5	. . .	. . .	1,131	452.48	21.70	56
06101	6	. . .	. . .	528	435.96	9.97	56
OVRHD					435.96	77.49	1136

Table A1(Cont.).

PROG	P R	PAGES OUT	PAGES IN	OTHER I/O	ELAPSED TIME	PROCESS TIME	WRKG SET
01101	1	. . .	. . .	10,600	429.87	83.78	62
02011	2	. . .	. . .	6,343	429.87	116.74	60
03001	3	. . .	. . .	3,766	431.93	68.97	56
04000	4	. . .	. . .	2,370	431.93	44.36	58
05110	5	. . .	. . .	1,149	431.93	21.88	58
06101	6	. . .	. . .	547	431.93	10.11	58
07100	7	. . .	. . .	177	411.27	3.90	56
OVRHD					411.27	73.09	1136
01101	1	. . .	. . .	10,284	419.55	81.35	66
02011	2	. . .	. . .	6,115	419.55	112.55	58
03001	3	. . .	. . .	3,648	421.62	67.05	56
04000	4	. . .	. . .	2,317	421.62	43.35	58
05110	5	. . .	. . .	1,121	421.62	21.36	58
06101	6	. . .	. . .	557	421.62	10.43	56
07100	7	. . .	. . .	216	421.62	4.70	58
08000	8	. . .	. . .	75	351.39	1.57	54
OVRHD					351.39	62.32	1138

Table A2. Non-paging Runs Repl I - Calculated

PROG	CPU UTIL	CYCLE TIME	TI/O RATE	CPU RATE	CPU QUANTUM	PAGEIN RATE	I/O RATE	CYCLE FACTOR
01101	0.33	0.025	39.74	122.22	0.00818	. . .	39.74	1.000
OVRHED	.11							
01101	.28	.029	35.05	123.76	.00808	. . .	35.05	0.997
02011	.34	.054	18.50	53.94	.01854	. . .	18.50	0.992
OVRHED	.16							
01101	.27	.030	33.32	124.46	.00804	. . .	33.32	0.996
02011	.29	.064	15.60	54.25	.01843	. . .	15.60	0.992
03001	.15	.126	7.92	53.61	.01865	. . .	7.92	0.987
OVRHED	.17							
01101	.22	.037	27.14	125.05	.00800	. . .	27.14	0.995
02011	.29	.064	15.57	54.19	.01845	. . .	15.57	0.992
03001	.16	.116	8.66	54.42	.01838	. . .	8.66	0.989
04000	.09	.200	5.01	53.33	.01875	. . .	5.01	0.985
OVRHED	.17							
01101	.21	.038	26.63	126.03	.00794	. . .	26.63	0.995
02011	.27	.067	14.86	54.30	.01842	. . .	14.86	0.992
03001	.16	.115	8.72	54.56	.01833	. . .	8.72	0.989
04000	.10	.188	5.33	53.09	.01884	. . .	5.33	0.986
05110	.04	.438	2.28	51.97	.01924	. . .	2.28	0.983
OVRHED	.18							
01101	.20	.040	25.30	126.00	.00794	. . .	25.30	0.995
02011	.27	.068	14.75	54.33	.01841	. . .	14.75	0.992
03001	.16	.115	8.71	54.58	.01832	. . .	8.71	0.989
04000	.10	.183	5.46	53.23	.01879	. . .	5.46	0.986
05110	.05	.400	2.50	52.17	.01917	. . .	2.50	0.983
06101	.02	.824	1.21	53.05	.01885	. . .	1.21	0.982
OVRHED	.18							

Table A2(Cont.).

PROG	CPU UTIL	CYCLE TIME	TI/O RATE	CPU RATE	CPU QUANTUM	PAGEIN RATE	I/O RATE	CYCLE FACTOR
01101	.20	.041	24.66	126.53	.00790	. . .	24.66	0.995
02011	.27	.068	14.76	54.34	.01840	. . .	14.76	0.992
03001	.16	.115	8.72	54.62	.01831	. . .	8.72	0.989
04000	.10	.182	5.49	53.44	.01871	. . .	5.49	0.987
05110	.05	.376	2.66	52.57	.01902	. . .	2.66	0.985
06101	.02	.788	1.27	54.22	.01844	. . .	1.27	0.983
07100	.01	2.311	0.43	45.63	.02192	. . .	0.43	0.982
OVRHED	.18							
01101	.19	.041	24.52	126.43	.00791	. . .	24.52	0.995
02011	.27	.069	14.58	54.34	.01840	. . .	14.58	0.992
03001	.16	.116	8.66	54.43	.01837	. . .	8.66	0.989
04000	.10	.182	5.50	53.47	.01870	. . .	5.50	0.986
05110	.05	.376	2.66	52.53	.01904	. . .	2.66	0.984
06101	.03	.756	1.32	53.48	.01870	. . .	1.32	0.982
07100	.01	1.943	0.52	46.20	.02165	. . .	0.52	0.981
08000	0.00	4.624	0.22	48.44	0.02065	. . .	0.22	0.981
OVRHED	.18							

Table A3. Non-paging Runs Repl II - Measured

PROG	P R	PAGES OUT	PAGES IN	OTHER I/O	ELAPSED TIME	PROCESS TIME	WRKG SET
01101	1	. . .	. . .	11,977	301.03	97.70	66
OVRHD					301.03	33.94	1140
01101	1	. . .	. . .	11,982	342.51	96.78	64
02011	2	. . .	. . .	6,388	344.58	117.73	58
OVRHD					344.51	53.45	1140
01101	1	. . .	. . .	11,965	359.20	96.29	64
02011	2	. . .	. . .	5,606	361.27	103.00	58
03001	3	. . .	. . .	2,856	357.14	52.76	56
OVRHD					357.14	59.86	1140
01101	1	. . .	. . .	11,838	431.71	94.14	66
02011	2	. . .	. . .	6,810	431.71	124.76	58
03001	3	. . .	. . .	3,736	433.77	68.35	56
04000	4	. . .	. . .	2,151	431.71	40.10	58
OVRHD					431.79	74.73	1140
01101	1	. . .	. . .	11,724	444.15	93.30	66
02011	2	. . .	. . .	6,571	444.15	120.78	58
03001	3	. . .	. . .	3,858	444.15	70.55	56
04000	4	. . .	. . .	2,391	444.15	44.88	58
05110	5	. . .	. . .	1,042	435.89	19.99	58
OVRHD					435.89	77.16	1140
01101	1	. . .	. . .	11,582	460.81	91.96	66
02011	2	. . .	. . .	6,751	460.81	123.82	58
03001	3	. . .	. . .	4,075	460.81	74.41	56
04000	4	. . .	. . .	2,523	460.81	47.29	58
05110	5	. . .	. . .	1,183	460.81	22.64	56
06101	6	. . .	. . .	544	448.41	10.13	56
OVRHD					448.41	79.96	1140
01101	1	. . .	. . .	11,159	450.57	88.46	66
02011	2	. . .	. . .	6,607	450.57	121.88	58
03001	3	. . .	. . .	3,948	450.57	72.14	60
04000	4	. . .	. . .	2,441	450.57	45.90	60
05110	5	. . .	. . .	1,169	450.57	22.16	58
06101	6	. . .	. . .	585	450.57	10.86	56
07100	7	. . .	. . .	198	429.92	4.21	56
OVRHD					429.92	76.83	1140
01101	1	. . .	. . .	10,327	427.90	82.02	66
02011	2	. . .	. . .	6,182	427.90	113.20	58
03001	3	. . .	. . .	3,785	427.90	69.14	58
04000	4	. . .	. . .	2,380	427.90	44.21	58
05110	5	. . .	. . .	1,161	427.90	21.99	56
06101	6	. . .	. . .	596	427.90	11.09	56
07100	7	. . .	. . .	228	425.83	4.92	56
08000	8	. . .	. . .	81	354.72	1.59	56
OVRHD					354.72	63.14	1140

Table A4. Non-paging Runs Repl II - Calculated

PROG	CPU UTIL	CYCLE TIME	TI/O RATE	CPU RATE	CPU QUANTUM	PAGEIN RATE	I/O RATE	CYCLE FACTOR
01101	0.33	0.025	39.79	122.60	0.00816	. . .	39.79	1.000
OVRHED	.11							
01101	.28	.029	34.99	123.81	.00808	. . .	34.99	0.997
02011	.34	.054	18.54	54.27	.01843	. . .	18.54	0.992
OVRHED	.16							
01101	.27	.030	33.31	124.27	.00805	. . .	33.31	0.996
02011	.29	.064	15.52	54.44	.01837	. . .	15.52	0.992
03001	.15	.125	8.00	54.15	.01847	. . .	8.00	0.987
OVRHED	.17							
01101	.22	.036	27.42	125.76	.00795	. . .	27.42	0.995
02011	.29	.063	15.78	54.59	.01832	. . .	15.78	0.992
03001	.16	.116	8.62	54.67	.01829	. . .	8.62	0.988
04000	.09	.201	4.99	53.66	.01864	. . .	4.99	0.985
OVRHED	.17							
01101	.21	.038	26.40	125.68	.00796	. . .	26.40	0.995
02011	.27	.068	14.80	54.41	.01838	. . .	14.80	0.992
03001	.16	.115	8.69	54.70	.01828	. . .	8.69	0.989
04000	.10	.186	5.39	53.30	.01876	. . .	5.39	0.986
05110	.05	.418	2.39	52.18	.01916	. . .	2.39	0.983
OVRHED	.18							
01101	.20	.040	25.14	125.96	.00794	. . .	25.14	0.995
02011	.27	.068	14.65	54.53	.01834	. . .	14.65	0.992
03001	.16	.113	8.85	54.78	.01826	. . .	8.85	0.989
04000	.10	.183	5.48	53.35	.01874	. . .	5.48	0.986
05110	.05	.389	2.57	52.31	.01912	. . .	2.57	0.983
06101	.02	.823	1.22	53.80	.01859	. . .	1.22	0.982
OVRHED	.18							
01101	.20	.040	24.77	126.16	.00793	. . .	24.77	0.995
02011	.27	.068	14.67	54.21	.01845	. . .	14.67	0.992
03001	.16	.114	8.77	54.74	.01827	. . .	8.77	0.989
04000	.10	.185	5.42	53.20	.01880	. . .	5.42	0.986
05110	.05	.385	2.60	52.80	.01894	. . .	2.60	0.983
06101	.02	.769	1.30	53.97	.01853	. . .	1.30	0.982
07100	.01	2.160	0.46	47.25	.02117	. . .	0.46	0.981
OVRHED	.18							
01101	.19	.041	24.14	125.92	.00794	. . .	24.14	0.995
02011	.27	.069	14.45	54.62	.01831	. . .	14.45	0.992
03001	.16	.113	8.85	54.76	.01826	. . .	8.85	0.989
04000	.10	.180	5.56	53.85	.01857	. . .	5.56	0.986
05110	.05	.368	2.72	52.83	.01893	. . .	2.72	0.984
06101	.03	.717	1.40	53.83	.01858	. . .	1.40	0.982
07100	.01	1.860	0.54	46.55	.02149	. . .	0.54	0.981
08000	0.00	4.326	0.23	51.74	0.01933	. . .	0.23	0.981
OVRHED	.18							

Table A5. Paging Runs Repl I - Measured

PROG	P R	PAGES OUT	PAGES IN	OTHER I/O	ELAPSED TIME	PROCESS TIME	WRKG SET
01101	1	41	160	12,002	441.10	95.26	110
02011	2	14	68	7,015	445.16	129.29	54
03001	3	2	3	3,909	447.19	71.72	42
04000	4	2	4	2,331	447.19	43.69	52
OVRHD		10	31		441.10	72.11	120
01101	1	95	296	12,002	461.23	95.29	66
02011	2	39	120	6,830	467.54	125.87	50
03001	3	28	34	4,119	467.54	75.22	44
04000	4	34	13	2,593	467.54	48.51	46
05110	5	73	111	1,081	467.54	20.71	60
OVRHD		26	93		461.23	78.74	62
01101	1	150	476	12,002	500.05	95.63	54
02011	2	132	336	6,983	508.69	128.51	46
03001	3	66	159	4,348	517.17	79.51	48
04000	4	141	163	2,763	517.17	51.76	46
05110	5	448	491	1,149	517.17	21.97	48
06101	6	569	1,090	261	517.17	5.32	36
OVRHD		97	341		500.05	94.21	34
01101	1	197	1,261	12,002	597.96	96.44	58
02011	2	253	987	7,288	609.39	135.87	46
03001	3	347	949	4,062	584.91	75.56	46
04000	4	458	954	2,581	611.67	49.02	44
05110	5	826	2,111	429	611.67	9.05	40
06101	6	732	1,751	178	611.67	3.67	26
07100	7	590	1,390	100	611.67	2.38	22
OVRHD		237	648		584.91	127.99	34
01101	1	1,392	10,734	11,921	2,183.56	99.97	50
02011	2	2,187	8,671	10,564	2,183.56	201.39	44
03001	3	. . .	. . .	. . .	. . .	. . .	. . .
04000	4	. . .	. . .	. . .	. . .	. . .	. . .
05110	5	3,225	8,202	525	2,185.59	12.55	26
06101	6	2,930	7,501	460	2,185.59	10.64	24
07100	7	2,691	6,921	367	2,185.59	8.97	22
08000	8	3,206	6,618	866	2,185.59	19.84	32
OVRHD		922	3,136		2,183.56	506.51	32

Table A6. Paging Runs Repl I - Calculated

PROG	CPU UTIL	CYCLE TIME	TI/O RATE	CPU RATE	CPU QUANTUM	PAGEIN RATE	I/O RATE	CYCLE FACTOR
01101	0.22	0.036	27.58	126.00	0.00783	0.4	27.21	0.995
02011	.29	.063	15.91	54.26	.01825	.2	15.76	0.992
03001	.16	.114	8.75	54.52	.01833	.0	8.74	0.988
04000	.10	.191	5.22	53.37	.01871	.0	5.22	0.985
OVRHED	.16					.1		0.994
01101	.21	.038	26.67	125.97	.00775	.6	26.02	0.995
02011	.27	.067	14.87	54.27	.01811	.3	14.61	0.992
03001	.16	.113	8.89	54.78	.01811	.1	8.81	0.988
04000	.10	.179	5.58	53.47	.01861	.0	5.55	0.985
05110	.04	.392	2.552	52.25	.01736	.2	2.31	0.982
OVRHED	.17					.2		0.993
01101	.19	.040	24.96	125.52	.00766	1.0	24.00	0.994
02011	.25	.069	14.39	54.35	.01756	.7	13.73	0.991
03001	.15	.115	8.72	54.70	.01764	.3	8.41	0.988
04000	.10	.177	5.66	53.40	.01769	.3	5.35	0.985
05110	.04	.315	3.17	52.35	.01339	.9	2.22	0.983
06101	.01	.383	2.61	49.26	.00393	2.1	0.51	0.981
OVRHED	.19					.7		0.992
01101	.16	.045	22.18	124.46	.00727	2.1	20.07	0.991
02011	.22	.074	13.58	53.65	.01642	1.6	11.96	0.989
03001	.13	.117	8.57	53.77	.01508	1.6	6.95	0.986
04000	.08	.173	5.78	52.68	.01386	1.6	4.22	0.984
05110	.02	.241	4.15	47.50	.00356	3.5	0.70	0.982
06101	.01	.317	3.16	48.80	.00190	2.9	0.29	0.980
07100	.00	.410	2.44	42.44	.00160	2.3	0.17	0.978
OVRHED	.22					1.1		0.989
01101	.05	.096	10.38	119.26	.00441	4.9	4.92	. . .
02011	.09	.114	8.81	52.46	.01047	4.0	4.84	. . .
03001	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
04000	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
05110	.01	.250	4.00	41.93	.00144	3.8	0.24	. . .
06101	.01	.275	3.64	43.32	.00134	3.4	0.21	. . .
07100	.00	.300	3.34	41.01	.00123	3.2	0.17	. . .
08000	.01	.292	3.43	43.69	0.00265	3.0	0.40	. . .
OVRHED	0.23					1.4		

Table A7. Paging Runs Repl II - Measured

PROG	P R	PAGES OUT	PAGES IN	OTHER I/O	ELAPSED TIME	PROCESS TIME	WRKG SET
01101	1	45	166	12,002	441.43	95.46	100
02011	2	16	85	6,919	443.54	127.67	64
03001	3	0	3	3,916	445.57	71.89	44
04000	4	21	7	2,295	445.57	43.00	54
OVRHD		18	52		441.43	72.22	110
01101	1	116	302	12,002	462.56	95.49	70
02011	2	63	170	6,859	468.95	126.63	52
03001	3	24	34	4,160	471.04	76.01	40
04000	4	37	20	2,610	471.04	48.80	44
05110	5	81	86	1,110	471.04	21.22	62
OVRHD		36	95		462.56	78.78	62
01101	1	150	459	12,002	487.47	95.54	56
02011	2	97	252	6,953	502.52	128.55	50
03001	3	31	86	4,242	504.88	78.36	42
04000	4	112	88	2,838	504.88	53.16	46
05110	5	388	378	1,173	504.88	22.46	48
06101	6	531	1,067	249	504.88	4.89	38
OVRHD		74	282		487.47	89.78	34
01101	1	234	1,556	12,002	644.82	96.51	56
02011	2	298	1,212	7,444	654.00	138.82	46
03001	3	402	1,156	4,180	627.30	78.19	44
04000	4	521	1,151	2,770	656.28	52.67	44
05110	5	951	2,336	537	656.28	11.176	44
06101	6	807	1,954	183	656.28	3.81	26
07100	7	629	1,556	91	656.28	2.30	20
OVRHD		255	755		627.297	138.81	34

Table A8. Paging Runs Repl II - Calculated

PROG	CPU UTIL	CYCLE TIME	TI/O RATE	CPU RATE	CPU QUANTUM	PAGEIN RATE	I/O RATE	CYCLE FACTOR
01101	0.22	0.036	27.57	125.74	0.00784	0.4	27.19	0.995
02011	.29	.063	15.79	54.20	.01823	.2	15.60	0.992
03001	.16	.114	8.80	54.48	.01834	.0	8.79	0.988
04000	.10	.193	5.17	53.40	.01867	.0	5.15	0.985
OVRHED	.16					.1		0.994
01101	.21	.038	26.60	125.71	.00776	.7	25.95	0.995
02011	.27	.067	14.99	54.17	.01801	.4	14.63	0.992
03001	.16	.112	8.91	54.75	.01812	.1	8.83	0.988
04000	.10	.179	5.59	53.51	.01855	.0	5.54	0.985
05110	.05	.394	2.54	52.37	.01772	.2	2.36	0.982
OVRHED	.17					.2		0.993
01101	.20	.039	25.57	125.63	.00796	0.9	24.62	0.994
02011	.26	.070	14.34	54.10	.01784	.5	13.84	0.991
03001	.16	.117	8.57	54.15	.01810	.2	8.40	0.988
04000	.11	.172	5.80	53.41	.01816	.2	5.62	0.985
05110	.04	.325	3.07	52.28	.01447	.7	2.33	0.983
06101	.01	.383	2.61	51.17	.00371	2.1	0.50	0.981
OVRHED	.18					.6		0.992
01101	.15	.048	21.03	124.38	.00712	2.4	18.61	0.991
02011	.21	.076	13.24	53.63	.01604	1.9	11.38	0.989
03001	.13	.118	8.51	53.47	.01465	1.8	6.67	0.987
04000	.08	.167	5.98	52.62	.01343	1.8	4.22	0.984
05110	.02	.228	4.38	48.14	.00389	3.6	0.82	0.983
06101	.01	.307	3.26	48.29	.00178	3.0	0.28	0.981
07100	.00	.398	2.51	40.00	0.00140	2.4	0.14	0.979
OVRHED	0.22					1.2		0.989

Table A9. Page Allocation Runs - Measured

PROG	P R	PAGES OUT	PAGES IN	OTHER I/O	ELAPSED TIME	PROCESS TIME	WRKG SET
CNTL	1	28	143	12,144	761.46	222.39	82
CNTL	2	23	73	9,123	761.46	164.80	58
CNTL	3	33	110	6,223	762.73	113.59	64
CNTL	4	57	275	3,409	762.73	64.75	78
OVRHED					761.46	110.71	
CNTL	1	70	263	8,101	655.25	142.30	66
CNTL	2	90	269	6,679	655.25	116.44	52
CNTL	3	83	210	5,353	655.25	94.19	52
CNTL	4	143	326	3,904	649.05	70.22	52
CNTL	5	184	381	3,975	636.48	74.19	54
OVRHED					636.48	95.19	
CNTL	1	312	1,534	3,785	393.59	68.75	62
CNTL	2	312	1,502	2,712	393.59	49.12	52
CNTL	3	419	1,554	1,848	393.59	33.66	52
CNTL	4	600	1,746	1,144	393.58	21.24	48
CNTL	5	685	1,854	538	391.53	10.87	38
CNTL	6	. . .	. . .	. . .	. . .	. . .	26
OVRHED					391.53	91.72	
CNTL	1	575	3,306	3,647	609.72	68.90	56
CNTL	2	579	2,787	3,161	609.72	60.90	52
CNTL	3	793	2,682	1,693	609.72	32.77	48
CNTL	4	915	2,500	711	609.72	14.02	40
CNTL	5	970	2,411	333	606.47	6.82	36
CNTL	6	901	2,192	194	606.47	4.33	26
CNTL	7	805	1,095	156	584.38	3.54	24
OVRHED					584.38	138.50	

Table A10. Page Allocation Runs - Calculated

PROG	CPU UTIL	CYCLE TIME	TI/O RATE	CPU RATE	CPU QUANTUM	PAGEIN RATE	I/O RATE	CYCLE FACTOR
CNTL 1	0.29	0.062	16.17	54.61	0.01801	0.2	15.95	0.996
CNTL 2	.22	.083	12.08	55.37	.01792	.1	11.98	.994
CNTL 3	.15	.120	8.30	54.78	.01794	.1	8.16	.993
CNTL 4	.08	.207	4.83	52.66	.01757	.4	4.47	.992
OVRHED	.15							
CNTL 1	.22	.078	12.77	56.94	.01701	.4	12.37	.994
CNTL 2	.18	.094	10.60	57.37	.01677	.4	10.20	.993
CNTL 3	.14	.118	8.49	56.85	.01693	.3	8.17	.993
CNTL 4	.11	.153	6.52	55.60	.01660	.5	6.02	.992
CNTL 5	.12	.146	6.85	53.59	.01703	.6	6.25	.992
OVRHED	.15							
CNTL 1	.18	.074	13.52	55.07	.01292	3.9	9.62	.989
CNTL 2	.13	.093	10.71	55.23	.01165	3.8	6.89	.988
CNTL 3	.09	.116	8.65	54.93	.00989	3.9	4.70	.987
CNTL 4	.05	.136	7.35	53.90	.00735	4.4	2.91	.987
CNTL 5	.03	.164	6.11	49.49	.00455	4.7	1.37	.986
CNTL 6	.01	. . .	4.20	. . .	. . .	3.9	0.31	.985
OVRHED	.23							
CNTL 1	.11	.088	11.40	52.95	.00990	5.4	5.98	.997
CNTL 2	.10	.103	9.76	51.92	.00964	4.6	5.19	.996
CNTL 3	.05	.139	7.18	51.69	.00749	4.4	2.78	.996
CNTL 4	.02	.190	5.27	50.77	.00437	4.1	1.17	.995
CNTL 5	.01	.221	4.52	48.95	.00249	4.0	0.55	.995
CNTL 6	.01	.254	3.93	45.01	.00181	3.6	0.32	.995
CNTL 7	.01	0.286	3.50	44.40	0.00171	3.2	0.26	0.995
OVRHED	0.24							

Table A11. Program Estimation Runs - Measured

PROG	PR	PAGES OUT	PAGES IN	OTHER I/O	ELAPSED TIME	PROCESS TIME	WRKG SET
000	1	1	44	8,462	288.76	67.18	56
CTL	2	9	21	4,302	288.76	79.81	52
CTL	3	23	47	2,467	290.81	46.37	64
CTL	4	51	76	1,293	280.56	24.54	96
OVRHED		12	32		280.56	45.27	96
011	1	23	144	8,409	270.62	67.66	92
CTL	2	0	4	4,015	270.62	74.35	54
CTL	3	28	47	2,230	270.62	42.04	56
CTL	4	57	101	1,114	258.43	21.15	76
OVRHED		5	14		258.43	42.63	94
101	1	22	165	8,313	260.51	66.82	94
CTL	2	0	4	3,777	260.51	69.83	56
CTL	3	6	8	2,129	262.60	39.77	58
CTL	4	59	112	1,050	250.35	19.99	64
OVRHED		15	49		250.35	41.53	82
110	1	5	44	8,440	294.87	67.58	72
CTL	2	11	23	4,415	294.87	81.51	58
CTL	3	25	43	2,575	296.96	48.29	62
CTL	4	60	97	1,359	286.67	25.81	84
OVRHED		15	29		286.67	46.35	80
CTL	1	24	117	11,915	746.14	218.38	84
CTL	2	29	78	8,961	748.17	161.44	58
CTL	3	39	97	6,044	748.17	110.52	66
001	4	70	239	8,718	737.97	68.36	70
OVRHED		16	60		737.97	108.75	94
CTL	1	26	143	12,395	776.57	227.01	90
CTL	2	17	57	9,330	776.57	168.15	60
CTL	3	34	117	6,418	778.60	117.22	66
010	4	52	115	8,720	768.41	68.48	58
OVRHED		21	53		768.41	112.33	96
CTL	1	38	167	12,075	760.68	219.93	82
CTL	2	27	68	9,174	760.68	164.09	62
CTL	3	36	128	6,255	762.76	113.55	64
100	4	59	125	8,720	752.52	68.26	64
OVRHED		25	72		752.52	110.79	84
CTL	1	31	166	11,941	752.53	219.04	82
CTL	2	26	81	9,025	752.53	162.67	58
CTL	3	35	116	6,148	752.53	112.41	64
111	4	65	240	8,718	742.29	69.14	74
OVRHED		19	65		742.29	109.74	88

Table A12. Program Estimation Runs - Calculated

PROG	CPU UTIL	CYCLE TIME	TI/O RATE	CPU RATE	PAGE RATE	PAGEIN RATE	EXP FACTOR	ADJ-CPU RATE
000 1	0.23	0.034	29.46	125.98	0.2	0.2	0.995	126.58
CNTL 2	.28	.067	14.93	54.92	0.1	0.1	0.991	54.38
CNTL 3	.16	.116	8.65	53.23	0.2	0.2	0.990	53.79
CNTL 4	.09	.205	4.88	52.73	0.5	0.3	0.989	53.32
OVRHED	.16				0.2	0.1		
011 1	.25	.032	31.08	126.20	0.6	0.5	0.995	126.80
CNTL 2	.28	.067	15.83	54.02	0.0	0.0	0.991	54.51
CNTL 3	.16	.119	8.42	53.07	0.3	0.2	0.989	53.65
CNTL 4	.08	.213	4.71	52.71	0.6	0.4	0.988	53.32
OVRHED	.17				0.1	0.1		
101 1	.26	.031	32.57	124.50	0.7	0.6	0.995	125.07
CNTL 2	.27	.069	14.52	54.10	0.0	0.0	0.993	54.48
CNTL 3	.15	.123	8.14	53.57	0.1	0.0	0.991	54.05
CNTL 4	.08	.215	4.65	52.58	0.7	0.4	0.990	53.09
OVRHED	.17				0.3	0.2		
110 1	.23	.035	28.78	124.91	0.2	0.1	0.995	125.53
CNTL 2	.28	.066	15.05	54.18	0.1	0.1	0.992	54.64
CNTL 3	.16	.113	8.82	53.37	0.2	0.1	0.990	53.90
CNTL 4	.09	.197	5.08	52.55	0.5	0.3	0.989	53.14
OVRHED	.16				0.2	0.1		
CNTL 1	.29	.062	16.13	54.57	0.2	0.2	0.995	54.86
CNTL 2	.22	.083	12.08	55.52	0.1	0.1	0.993	55.91
CNTL 3	.15	.122	8.21	54.70	0.2	0.1	0.992	55.15
001 4	.09	.082	12.14	127.55	0.4	0.3	0.992	128.53
OVRHED	.15				0.1	0.1		
CNTL 1	.29	.062	16.15	54.61	0.2	0.2	0.995	54.90
CNTL 2	.22	.083	12.09	55.49	0.1	0.1	0.993	55.88
CNTL 3	.15	.119	8.39	54.76	0.2	0.2	0.992	55.21
010 4	.09	.087	11.50	127.36	0.2	0.2	0.992	128.35
OVRHED	.15				0.1	0.1		
CNTL 1	.29	.062	16.09	54.91	0.3	0.2	0.995	55.20
CNTL 2	.22	.082	12.15	55.91	0.1	0.1	0.993	56.31
CNTL 3	.15	.119	8.37	55.10	0.2	0.2	0.992	55.55
100 4	.09	.085	11.76	127.77	0.2	0.2	0.992	128.76
OVRHED	.15				0.1	0.1		
CNTL 1	.29	.062	16.09	54.52	0.3	0.2	0.995	54.82
CNTL 2	.22	.083	12.10	55.49	0.1	0.1	0.993	55.88
CNTL 3	.15	.120	8.33	54.70	0.2	0.2	0.992	55.16
100 4	.09	0.083	12.07	126.12	0.4	0.3	0.992	127.10
OVRHED	0.15				0.1	0.1		

SELECTED BIBLIOGRAPHY

SELECTED BIBLIOGRAPHY

1. Bard, Y. "Characterization of Program Paging in a Time-Sharing Environment", IBM Journal of Research and Development, Vol. 17, No. 5, (Sept 1973), pp. 387-393.

2. _____. "Application of the Page Survival Index (PSI) to Virtual-Memory System Performance", IBM Journal of Research and Development, Vol. 19, (May 1975), pp. 212-220.

3. _____. "A Characterization of VM/370 Workloads", IBM Cambridge Scientific Center, Technical Report No. G320-2111, (April 1976).

4. Baskett, F., and Gomez, F.P. "Processor Sharing in a Central Server Queuing Model of Multiprogramming with Applications", Sixth Annual Princeton Conference on Information Sciences and Systems, (1972), pp. 598-603.

5. Bass, L.J. "On Optimal Processor Scheduling for Multiprogramming", SIAM J. COMPUT., Vol. 2, No. 4, (December 1973), pp. 273-80.

6. Belady, L.A., and Kuehner, C.J. "Dynamic Space-Sharing in Computer Systems", Communications of the ACM, Vol. 12, No. 5, (May 1969), pp. 282-288.

7. Boyd, D.L. "A Multiple Resource Model for A Batch-Processing Multiprogramming System", National Technical Information Service, U.S. Department of Commerce, Report AD-722332, (March 1971).

8. Boyse, J.W., and Warn, D.R. "A Straightforward Model for Computer Perfcrmance Prediction", Computing Surveys, Vol. 7, No. 2, (June 1975), pp. 73-93.

9. Brandwajn, A. "A Queuing Model of Multiprogrammed Computer Systems Under Full Load Conditions", Communications of the ACM, (April 1977), pp. 222-240.

10. Buchholz, W. "A Synthetic Job for Measuring System Performance", IBM Systems Journal, Vol. 8, No. 4, (1969), pp. 309-318.

11. Burge, W.H., and Konheim, A.G. "An Accessing Model", Journal of the ACM, Vol. 18, No. 3, (July 1971), pp. 400-404.

12. Chamberlain, D.D., Fuller, S.H., and Liu, L.Y. An Analysis of Page Allocation Strategies for Multiprogramming Systems with Virtual Memory", IBM Journal of Research and Development, Vol. 17, No. 5, (Sept 1973), pp. 404-412.

13. Chang, W. "Single-Server Queuing Processes in Computing Systems", IBM Systems Journal, Vol. 9, No. 1, (1970), pp. 36-71.

14. Courtois, P.J. "Decomposibility, Instabilities, and Saturation in Multiprogramming Systems", Communications of the ACM, Vol. 18, No. 7, (July 1975), pp. 371-376.

15. Cox, D.R. "A Use of Complex Probabilities in the Theory of Stochastic Processes", Proceedings of the Cambridge Philosophical Society, Vol. 51, (1955), pp. 313-319.

16. _____. Planning of Experiments, New York: John Wiley and Sons,Inc., 1958.

17. Denning, P.J. "Effects of Scheduling on File Memory Operations", Spring Joint Computer Conference, (1967), pp. 9-21.

18. _____. "The Working Set Model for Program Behavior", Communications of the ACM, Vol. 11, No. 5, (May 1968), pp. 323-33.

19. Denning, P.J., and Schwartz, S.C. "Properties of the Working Set Model", Communications of the ACM, Vol. 15, No. 1, (March 1972), pp. 191-198.

20. Fenichel, R.R., and Grossman, A.J. "An Analytic Model of Multiprogrammed Computing", Proceedings of the Spring Joint Computer Conference, (1969), pp. 717-21.

21. Fernandez, E.B., and Lang, T. "Computation of Lower Bounds for Multiprocessor Schedules", IBM Journal of Research and Development, Vol. 19, No. 5, (September 1975), pp. 435-44.

22. Finn, J.D. A General Model for Multivariate Analysis, New York: Holt, Rinehart and Winston, Inc., 1974.

23. Forbes, K., and Goldsworthy, A.W. "A Prescheduling Algorithm -- Scheduling a Suitable Mix Prior to Processing", The Computer Journal, Vol. 20, No. 1, (1977), pp. 27-29.

24. Franklin, M.A., and Gupta, R.K. "Computation of Page Fault Probability from Program Transition Diagrams", Communications of the ACM, Vol. 17, No. 4, (April 1974), pp. 186-191.

25. Gaver, Jr., D.P. "A Waiting Line with Interrupted Service Including Priorities", Journal of the Royal Statistical Society, Series 13, No. 24, (1962), pp. 73-90.

26. _____. "Probability Models for Multiprogramming Computer Systems", Journal of the ACM, Vol. 14,No. 3, (1967), pp. 423-38.

27. Ghanem, M.Z. "Dynamic Partitioning of Main Memory Using the Working Set Concept", IBM Journal of Research and Development, Vol. 19, No. 5, (September 1975), pp. 445-50.

28. Gordon, W.J., and Newell, G.F. "Closed Queuing Systems with Exponential Servers", Operations Research, Vol. 15, No. 2, (April 1967), pp. 254-265.

29. IBM Corporation "Analysis of Some Queuing Models in Real-Time Systems",Form GF20-0007, Data Processing Division, White Plains, N.Y. 10604.

30. Isaacson, E., and Keller, H.B. Analysis of Numerical Methods, New York: John Wiley & Sons, Inc., 1966.

31. Iverson, K.E. A Programming Language, New York, John Wiley & Sons, Inc., 1962.

32. Jackson, J.R. "Jobshop-Like Queuing Systems", Management Science, Vol. 10, No. 1, (Oct 1963), pp. 131-142.

33. Kimbleton, S.R. "Batch Computer Scheduling: A Heuristically Motivated Approach", Office of Naval Research, Report AD-A007922 (September 1974).

34. Lewis, P.A.W., and Schedler, G.S. "A Cyclic Queue Model of Multiprogramming", Journal of the ACM, Vol. 18, No. 2, (April 1971), pp. 199-220.

35. Miller, Jr., R.G. "Priority Queues", The Annals of Mathematical Statistics, Vol. 31, (1960), pp. 86-103.

36. Muntz, R.L. "Poisson Departure Processes and Queuing Networks", Conference on Information Sciences and Systems, (March 1973), pp. 435-44C.

37. National Bureau of Standards, Fractional Factorial Designs for Factors at 2 Levels, Washington: U. S. Department of Commerce, 1957.

38. Oden, P.H., and Schedler, G.S. "A Model of Memory Contention in a Paging Machine", Communications of the ACM, Vol. 15, NO. 8, (August 1972), pp 761-771.

39. Reiser, M. "Interactive Modeling of Computer Systems", IBM Systems Journal, Vol. 15, No. 4, (1976), pp. 308-27.

40. Saltzer, J.H. "A Simple Linear Model of Demand Paging Performance", Communications of the ACM, Vcl. 17, No. 4, (April 1974), pp. 181-186.

41. Seaman, P.H., Lind, R.A., and Wilson, T.L. "On Teleprocessing System Design, Part IV: An Analysis of Auxiliary Storage Activity", IBM Systems Journal, Vol. 5, No. 3, (1968), pp. 158-170.

42. Sreenivasan, K., and Kleinman, A.J. "On the Construction of a Representative Synthetic Workload", Communications of the ACM, Vol. 17, No. 3, (March 1974), pp. 127-33.

43. Teorey, T.J. "Properties of Disk Scheduling Policies on Multiprogrammed Ccmputer Systems", Fall Joint Computer Conference, (1972), pp. 1-11.

44. Waters, S.W. "Estimating Magnetic Disk Seeks", The Computer Journal, Vcl. 18, No. 1, (1975), pp. 12-17.

45. Wilhelm, N.C. "A General Model for the Performance of Disk Systems", Journal of the ACM, Vol. 24, No. 1, (Jan 1977), pp. 14-31.

46. Welch, P.D. "On Pre-Emptive Resume Priority Queues", The Annals of Mathematical Statistics, Vol. 35, (1964), pp. 1340-48.