Computer Systems and Architecture
J. P. Hayes Editor

Models for Parallel Processing Within Programs: Application to CPU:I/O and I/O:I/O Overlap

D. Towsley, K.M. Chandy, and J.C. Browne
The University of Texas at Austin

Approximate queueing models for internal parallel processing by individual programs in a multiprogrammed system are developed in this paper. The solution technique is developed by network decomposition. The models are formulated in terms of CPU:I/O and I/O:I/O overlap and applied to the analysis of these problems. The percentage performance improvement from CPU:I/O overlap is found to be greatest for systems which are in approximate CPU:I/O utilization balance and for low degrees of multiprogramming. The percentage improvement from I/O:I/O overlap is found to be greatest for systems in which the I/O system is more utilized than the CPU.

Key Words and Phrases: multiprogramming, parallel processing, queueing network models, multiprocessing of computation and I/O

CR Categories: 4.32, 8.1

1. Introduction

This paper develops approximate queueing models for internal parallel processing by individual programs in a multiprogrammed system. The models are formulated in terms of the familiar CPU:I/O and I/O:I/O overlap problems, although they are generally applicable to the study of parallel processing. The accuracy and validity of the models are carefully analyzed. The models are then used to analyze the performance improvement to be expected from implementation of CPU:I/O and I/O:I/O overlap in multiprogrammed systems.

There is rising interest in the application of parallelism within programs. As mini and microcomputer systems become cheaper, there will be a growth of multiprogramming systems. In these circumstances, increasing levels of parallelism may have marked effects on performance measures. Further applications of parallelism may be found in recent database system design. Interest has arisen in the design of database systems in which complex queries are processed by being broken up into simple queries all to be processed in parallel. The most significant current application of parallelism is in CPU:I/O overlap in data processing applications on conventional computer systems. Most major operating systems offer the capability for a job to overlap its own I/O. These facilities are not commonly used. It is also the case that the performance benefits which may be derived, and the circumstances under which performance improvement may be obtained, are poorly understood. The study of parallel processing within individual programs has been hindered by the dearth of accurate models for processes which execute simultaneously on more than one processor. The queueing network model, the most widely used computer system modeling technology, is almost always formulated with the assumption that a program (job) executes serially on the several processors (queues, servers) of the model system. These models do not allow a program to hold more than one processor simultaneously. There have been two previous studies which have attempted to generalize network models to include parallel processing by an individual program within a multiprogrammed system. Peterson and Bulgren [18] construct Markov models which are applicable to systems including multiple buffering for several I/O stations (CPU:I/O overlap). The applicability of these models is limited to small systems because of the state space explosion problem. Price [19] develops models for multiple buffering of a single file per program and for the case where the file set of program shares a common pool of buffers. Price also analyzes the effect of CPU service-time distribution on the effectiveness of multiple buffering. Maekawa and Boyd [17] have modeled a number of cases of CPU:I/O overlap. These authors investigated the effects of service rate distributions and to a limited extent the effect of CPU scheduling disciplines. Related papers that treat parallel processing of CPU's in multi-CPU systems are [4] and [22].

This paper presents models for multiprogrammed systems in which programs may either partially or completely overlap CPU and I/O processing and where two I/O activities may partially or completely overlap among themselves and CPU processing. These model systems admit of exact solution for a class of systems and of inexact, but very accurate, solution for a wider class of models. The accuracy and validity of these models have been carefully verified by validation against detailed

Authors' address: Dept. of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712.

simulations and against actual operation of a multiprogrammed computer system.

The domains of degree of overlap, degree of multiprogramming, and system resource utilization where CPU:I/O and I/O:I/O overlap produce significant performance gains are determined. The percentage performance improvements are found to be greatest for systems which are in approximate CPU:I/O utilization balance for CPU:I/O overlap for very low degrees of multiprogramming, and for systems in which the I/O is heavily utilized for I/O:I/O overlap.

The structure of this paper is as follows. In Section 2, we summarize fundamental concepts in queueing network analysis including *central server models* (Figure 1) in *local balance*, and describe "*Norton's Theorem.*" In Section 3, we present a simple CPU:I/O overlapped processing structure for a single program and present a model for the analysis of central server networks populated by these programs. Section 4 presents a similar model for the overlap of two I/O and CPU activities. In Sections 6 and 7, we present comparisons between systems with and without either CPU:I/O or I/O:I/O overlap.

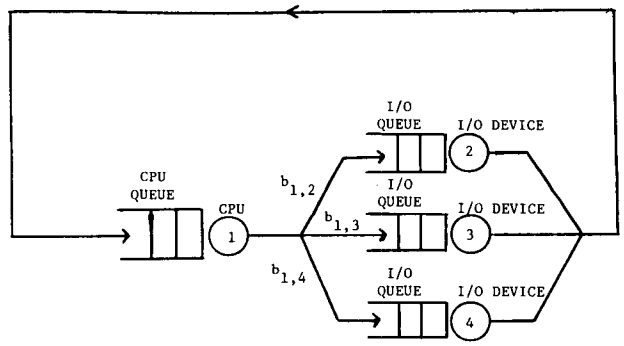
2. Local Balance and Norton's Theorem

The model solution techniques developed in this paper are based on three fundamental concepts: the central server model [1, 5], local balance [2, 6, 9] and "Norton's Theorem" [7]. A brief exposition of these concepts is included for ease of reading the subsequent discussion.

The central server model (Figure 1) is a commonly used queueing network representation of multiprogrammed computing systems (without CPU:I/O overlap). The CPU and I/O devices are represented explicitly as servers (circles in Figure 1) and jobs queue up in separate queues for the different servers. There is a fixed number of jobs in the system; this number is referred to as the degree of multiprogramming. Main memory is treated implicitly in the model by adjusting the degree of multiprogramming. A job may be waiting for, or receiving service from, *one* of the servers. When a job completes CPU service, it branches to an I/O device with a fixed probability, independent of the state of the system. The model is generally used to compute steady-state values for performance metrics such as device utilization, mean response times and queue length distributions. This simplistic representation of complex systems has been shown to be useful in several studies of real systems; see [3] or [20], for example. Perhaps more importantly, the model helps provide insight into the relationships between key performance parameters.

A central server network [5] is in local balance if (1) the branching probabilities depend only on the job class and I/O device; (2) the service disciplines are either first come first serve (FCFS), processor sharing (PS), infinite

Fig. 1. The central server model.



server (IS), or last come first serve preemptive resume (LCFSPR); (3) processors with the FCFS discipline have exponential service distributions with a mean independent of the job class; (4) processors with PS, IS, or LCFSPR disciplines have arbitrary differentiable distributions [9] which may differ between job classes. If the above criteria are met, the network has a steady-state solution of the product form [2, 9]. For a thorough discussion of systems which satisfy local balance, see [9]. Efficient computational procedures exist for product form solutions which make locally balanced networks easy to analyze [5].

One of the consequences of local balance is a theorem analogous to Norton's Theorem in electrical circuit theory [7]. The theorem allows the transformation of a local balance central server network into a two-queue network containing the original CPU and a "composite I/O" which represents all the effects of the I/O subsystem in the original model, Figure 2. The CPU steady-state thruput, mean wait time, and queue length distribution for the two-queue network remain identical to those of the original network. The transformation of the I/O subsystem into a composite I/O is independent of the CPU parameters. Thus a study of the original network under a variety of CPU parameters is easily done by studying the CPU/composite-I/O network. This attribute of simple parametric analysis of the CPU will also apply to our approximation models.

The models of this paper use only one job class; i.e. all jobs have the same service distributions and branching probabilities. Therefore we present a description of Norton's Theorem for a single job class. Assume there are N identical jobs. The composite I/O service rate at any time depends upon the number of jobs in the I/O subsystem (or equivalently the composite I/O queue) at that time. These composite I/O service rates can be determined by analyzing the original network when the CPU has been "shorted" (Figure 3). Shorting the CPU is accomplished by setting the CPU service time to zero. The composite I/O service rate when there are N jobs in the composite I/O queue is set to the thruput of the jobs through the shorted CPU when there are N jobs in the shorted CPU model. The CPU/composite-I/O model with the same CPU parameters as in the original model

Fig. 2. The CPU/composite-I/O system derived by applying Norton's Theorem.

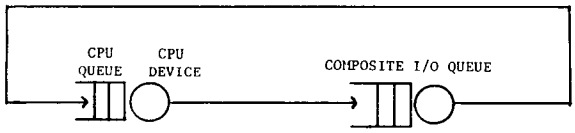
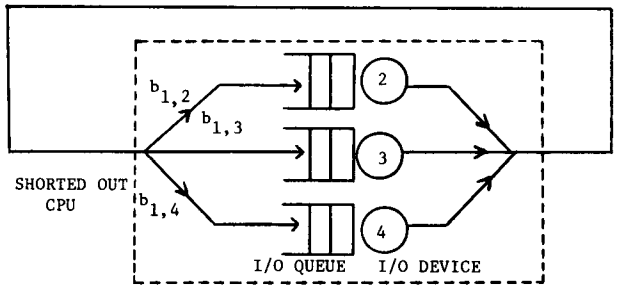


Fig. 3. Shorting out the CPU: applying Norton's Theorem.



and these queue length dependent composite I/O service rates produce CPU statistics identical to those of the original model.

Example. Consider a local balance central server network with three identical jobs and two I/O devices. Each job has equal probability of going to either I/O device. The mean service times for I/O devices 1 and 2 are 1.0 and 2.0 respectively. With the CPU shorted out and $n = 1, 2, 3$ jobs in the network, the thruput through the shorted CPU is 2/3, 6/7, and 14/15 respectively. The composite I/O with n jobs in the queue, $n = 1, 2, 3$, has service rates 2/7, 6/7, and 14/15 respectively, in the CPU/composite-I/O network. Both the original model and the CPU/composite-I/O model give identical equilibrium queue length and mean wait time statistics.

3. A CPU:I/O Overlap Model

The assumption normally made in a central server network model [5] is that a job alternates between CPU and I/O activities. The job may be thought of as repeating cycles, where each cycle (Figure 4) consists of two tasks: a task requiring use of the CPU followed by one requiring use of an I/O. In an overlapped system the cycle is more complex: for example (Figure 5), a cycle may consist of three tasks, two of which require the use of the CPU while the other requires an I/O; furthermore, one of the CPU tasks must be completed before the other two tasks can be initiated. We model the structure of the cycle by a single input, single output precedence graph. A cycle consisting of K tasks indexed 1, ... , K is represented by a graph with K vertices, where the ith vertex represents the ith task. The identity of the processor required by each task is written next to the corresponding vertex. If task j cannot be initiated until task i is completed, then there must be a directed path from i to j in the graph. There is a path from the input point to every

Fig. 4. Task precedence graph for nonoverlapped processing.

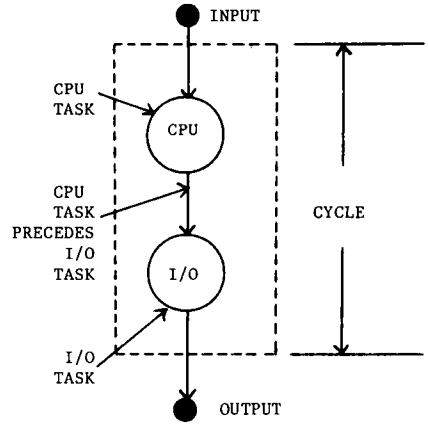
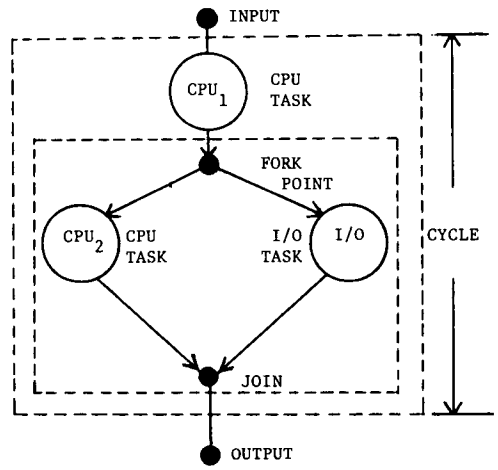


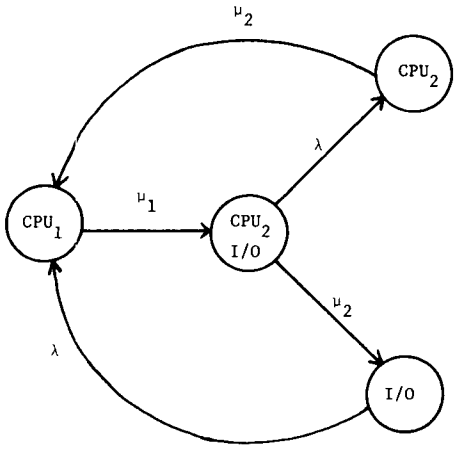
Fig. 5. Job model for CPU-I/O overlap.



vertex in the graph, and a path from every vertex to the output point. It is convenient to introduce *fork points* and *probabilistic branch points* into the graph. A vertex has exactly one edge leading out of it; this edge may terminate at another vertex or at a fork, probabilistic branch, or output point. Branch points and forks have two or more edges leading out of them. Associated with each edge leading from a probabilistic branch point is a probability and a distinct single input, single output subgraph shown in the diagrams in dotted lines. (The subgraph may consist of a single edge joining the input and output points.) When execution reaches a probabilistic branch point, one and only one of the output edges is selected (with the appropriate probability) and the subgraph associated with that edge must be processed; the subgraphs associated with all the other edges leading from the branch point are not processed. All probabilistic branch subgraphs have a common output vertex, and a common input vertex, viz. the probabilistic branch point.

After execution reaches a *fork* point it must proceed along each and every one of the edges leading out of the fork. We assume that we are given service time distributions for each task in the cycle and that service times

Fig. 6. State transitions of a single job in the CPU: I/O overlap case.



and branching probabilities are independent of the state of the system.

We depart from the sequential cycle structure of the usual central server model by allowing a job the capability of overlapping some of its CPU activity with I/O requests. A reasonable structure is to allow a job to have two CPU tasks (Figure 5), one which is not overlapped with any I/O (CPU_1) and a second task (CPU_2) which is overlapped with an I/O request. During CPU_1, the job partakes only in CPU activity, possibly setting up buffers for an impending I/O request. At the end of this task, the job initiates both an I/O request and the second CPU task, CPU_2. The second CPU task and the I/O may be processed in parallel. Only upon completion of *both* I/O and CPU_2 does the job revert to CPU_1 and repeat the cycle.

We assume, for ease of solution, that the service times for each task are exponentially distributed. The Markovian state transition diagram for one job with an overlapped processing structure is found in Figure 6. The mean service times for CPU_1, CPU_2, and I/O are $1/\mu_1$, $1/\mu_2$, and $1/\lambda$ respectively. A job may be in one of the following four states:

(1)	CPU_1	CPU executes first task (I/O idle)
(2)	CPU_2	CPU executes second task (I/O idle)
(3)	I/O	I/O executes (CPU idle)
(4)	CPU_2-I/O	CPU executes second task in parallel with I/O execution

The above model suffers one of the flaws of Price's models. It assumes that a job will attempt to overlap all its I/O activity with CPU_2. A more realistic model should allow a mixture of sequential and overlapped processing. Let p be the probability that a job, upon completion of CPU_1, requires overlapped processing of CPU_2 and I/O. Then, $\bar{p} = 1 - p$ is the probability that a job, upon completion of CPU_1, initiates I/O and relinquishes the CPU. The process state graph is found in Figure 7 and the Markov state transition diagram in Figure 8. We will refer to this as CPU:I/O overlapping. A model which exhibits overlapping generally does not

Fig. 7. A more realistic CPU: I/O overlap model.

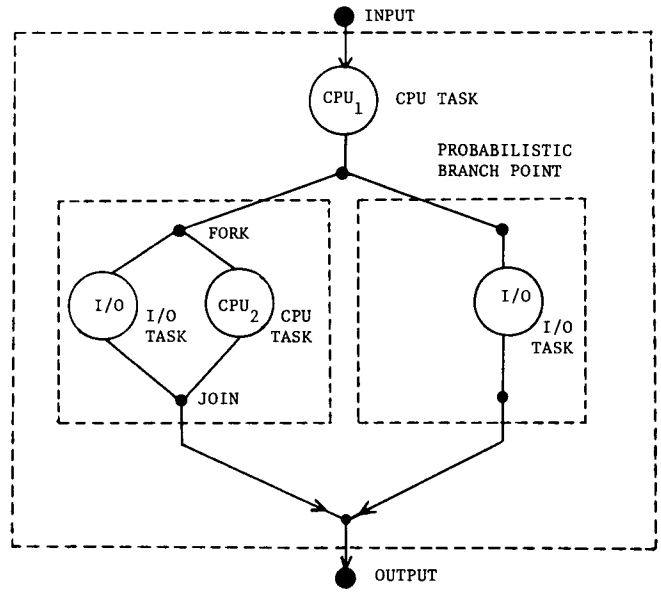
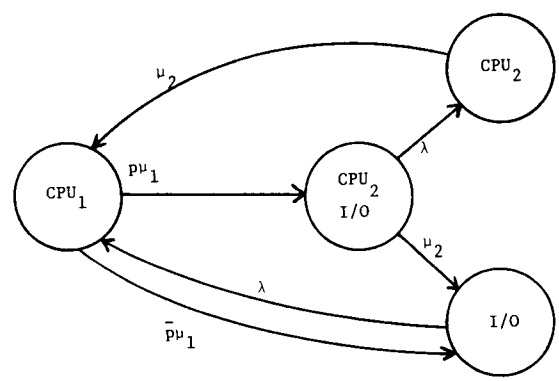


Fig. 8. State transitions for a single job with the realistic overlap model.



satisfy local balance, forcing us to resort to approximation methods.

A central server network with one I/O device and identical jobs exhibiting CPU:I/O overlapping can be solved exactly when the queueing disciplines are all FCFS. We can apply a recursive technique similar to that of Herzog, Woo, and Chandy [12] to this two-queue model. This technique provides CPU utilization U_0, CPU thruput T_0, mean CPU queue length $\bar{q}$, and mean CPU wait time $\bar{w}$. The details of the two-queue analysis may be found in [25].

A central server network with an arbitrary number of I/O devices and identical jobs each with CPU:I/O processing can be solved approximately by reducing it to a two-queue network via Norton's Theorem and then using the above techniques. The details are found in the following algorithm.

Algorithm

Step 1. *Obtain composite I/O service rates.* Consider the given central server model. Construct the shorted CPU model in which there is no overlap, all I/O times are exponential, and the CPU service time is set to zero. This shorted CPU model satisfies

local balance and is easily analyzed. Determine queue dependent, composite I/O service rates by analyzing the shorted CPU model.

Step 2. *Solve the two-queue, CPU/composite-I/O model exactly via recursive techniques.* The processing structure and CPU parameters are identical to those of the original model. The composite-I/O parameters are specified by Step 1. This completely specifies the two-queue model. Analyze this model to determine U_0, T_0, $\bar{q}$, $\bar{w}$.

Example. Consider the two I/O models from the first example. Assume the jobs have processing structures identical to that in Figure 5 with the means for CPU_1, CPU_2 each 1.

Step 1. The composite I/O service rates (from Section 2) when there are n jobs, $n = 1, 2, 3$ in the composite I/O queue are 2/3, 6-7, and 14/15 respectively.

Step 2. We now have a two-queue model. This model is solved using the techniques presented in [T1] to obtain $U_0 = 0.8$, $T_0 = 0.8$, $\bar{q} = 1.2$, and $\bar{w} = 1.6$.

4. An I/O:I/O Overlap Model

We consider jobs which may initiate two consecutive I/O requests before relinquishing the CPU. We assume the processing structure illustrated in Figure 9. There are two CPU tasks subscripted 1 and 2 in this structure. At the end of CPU_1, the job initiates one I/O request and may start the second CPU task (CPU_2). At the end of CPU_2, the job initiates a second I/O request and relinquishes the CPU. Only upon completion of all I/O requests, does the job reenter CPU_1 and repeat the cycle. This is illustrated by the fork and join subgraph in Figure 9. We assume a job initiates two parallel I/O requests (and enters the subgraph) with probability p. The other branch in Figure 9 is traversed when the job initiates only one I/O request. This occurs with probability $\bar{p} = 1 - p$. We will refer to this as I/O:I/O overlapping.

If the times for each task are exponential, the behavior of the job is governed by a Markov process. The Markov state transition diagram for the job running alone is illustrated in Figure 10. In this case the mean service times for CPU_1, CPU_2, and I/O are $1/\mu_1$, $1/\mu_2$, and $1/\lambda$ respectively. The state space includes: three no-overlap states (CPU_1, CPU_2, I/O), the state in which the job is processing two I/O requests (2 I/O), and the state in which the job is processing CPU_2 and I/O (CPU_2-I/O).

A central server network with one I/O device and in which all jobs have the same precedence graph can be solved exactly when the CPU queueing discipline is FCFS and the I/O discipline is PS. The solution technique is similar to that of Herzog, Chandy, and Woo [12] and may be found in [25].

A central server network with an arbitrary number of I/O devices, customers with identical I/O:I/O processing and all FCFS queueing disciplines may be solved approximately. The technique is similar to that of the algorithm in Section 3 and consists of using Norton's Theorem to reduce the network to a two-queue network,

Fig. 9. Model of a job with I/O: I/O overlap.

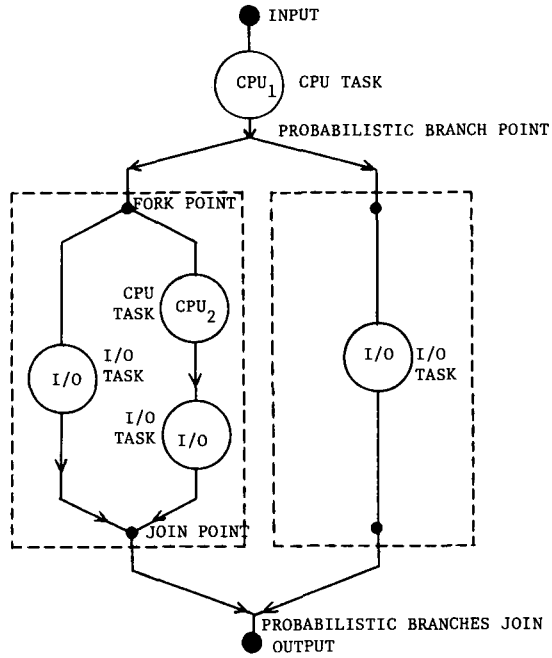
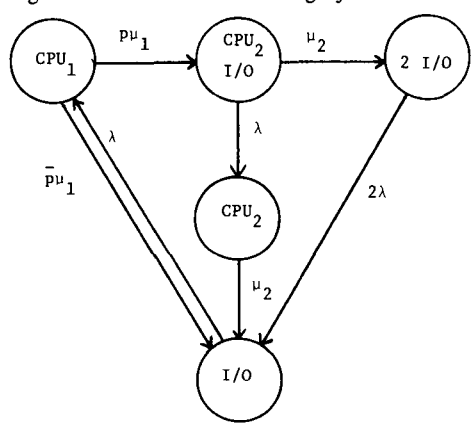


Fig. 10. State transitions of a single job with I/O: I/O overlap.



where the PS discipline is used by the composite-I/O. The resulting two-queue network is solved using algorithms in [25] to determine U_0, T_0, $\bar{q}$, and $\bar{w}$ for the CPU.

5. Validation of Overlapped Processing Models

To validate the queueing model we compared the performance measures predicted by the model with those obtained from simulation and from measurements made on a real system running a synthetic job load. We shall discuss the simulator now and the measurements later. We wrote a simulator to model arbitrary central server networks with CPU:I/O or I/O:I/O overlap, using the simulation language Aspol [11].

We compared the CPU utilization, mean queue length, and mean wait time for the analytic and simulation models. These comparisons are couched in terms of

two parameters: error tolerance z and level of confidence p. These parameters are defined later. We first determined values U_0, q_0, and w_0 for the utilization, mean queue length, and mean wait time respectively, obtained from the queueing model.

We then made several independent runs of the simulator. We initially experimented with the length of each run to ensure that the runs were long enough to yield credible results. Results from Smith [24] and Lavenberg [15] on the statistics of Markovian processes allow us to assume that the point estimates for the performance metrics obtained from the simulation have the *t-distribution* [16]. The parameters of this distribution are computed from the point estimates obtained from the independent runs. From this distribution we computed the probability $P_u(z)$ that the point estimate for the CPU utilization for any given run will lie in the interval $[U_0/(1 + z), U_1/(1 - z)]$ where z is the tolerance; in our experiments, z was usually set to 0.05. We also computed $p_q(z)$, the probability that the point estimate for mean CPU queue length for any given run will lie in the interval $[q_0/(1 + Nz), q_0/(1 - Nz)]$ where N is the degree of multiprogramming (i.e. the job population). We also computed $p_w(z)$, the probability that the mean wait time obtained from a simulation run will be in the interval $[W_0/(1 + T_0z), W_0/(1 - T_0z)]$ where T_0 is the mean cycle time obtained from the analytic model. Define $p(z) = \min[p_u(z), p_q(z), p_w(z)]$. We postulate the hypothesis that the analytic model is within tolerance $z = 0.05$ of the simulation model with probability $p(z) \geq 0.90$. This definition of tolerance was motivated by a similar definition in a paper by Chandy et al. [8].

We ran 36 CPU:I/O overlapped processing models to test the hypothesis. In general these models were fairly well balanced, though several were either strongly CPU-bound or I/O-bound. Most models were constructed to show strong overlap patterns. For $z = 0.05$, the hypothesis fit all the models with $p(z) \geq 0.90$. A nonoverlap (Sauer–Chandy) model [21], for the same hypothesis, fit only 16 of the 36 models with $p(0.05) \geq 0.90$.

The nonoverlap model was parameterized in the following way. The mean and coefficient of variation for the CPU service time requirements for each I/O request were calculated from the overlap model CPU time requirements and used to construct a multiexponential stage CPU service time distribution for the Sauer–Chandy nonoverlap model. All other parameters were set equal to those for the overlap model. Table I presents some results for three models.

We also ran 60 I/O:I/O overlapped processing models. Again, most were well balanced, but some were either strongly CPU or I/O bound. Most were constructed to show strong overlap behavior. For $z = 0.05$, the hypothesis fits all but one model with $p \geq 0.90$. For $z = 0.06$, the hypothesis fits the deviant model with $p \geq 0.90$. The nonoverlapped model, for the same hypothesis, fit only 13 of the 60 cases with $p \geq 0.90$. Table II presents results for three models.

Table I. CPU: I/O overlap.

Number of Customers	2	4	8
CPU utilization (approximation model)	0.54	0.74	0.90
(simulation model)	0.52	0.73	0.93
(nonoverlap model)	0.46	0.68	0.88
CPU mean queue length (approximation model)	0.78	1.67	3.84
(simulation model)	0.72	1.57	3.94
CPU mean wait time (ms) (approximation model)	14.60	22.70	42.60
(simulation model)	13.60	21.30	43.00
CPU_1 mean service (ms)	0.00		
CPU_2 mean service (ms)	10.00		
Probability of overlap	1.00		
I/O 1 mean service (ms)	6.00		
Branching probability	0.10		
I/O 2 mean service (ms)	7.50		
Branching probability	0.10		
I/O 3 mean service (ms)	1.50		
Branching probability	0.50		
I/O 4 mean service (ms)	2.00		
Branching probability	0.30		

Table II. I/O:I/O overlap.

Number of Customers	2	4	8
CPU utilization (approximation model)	0.470	0.67	0.85
(simulation model)	0.460	0.67	0.86
(nonoverlap model)	0.400	0.59	0.76
CPU mean queue length (approximation model)	0.600	1.29	2.99
(simulation model)	0.590	1.29	3.01
CPU mean wait time (ms) (approximation model)	12.800	19.10	35.00
(simulation model)	12.700	18.30	35.50
CPU_1 mean service (ms)	10.000		
CPU_2 mean service (ms)	0.000		
Probability of overlap	1.000		
I/O 1 mean service (ms)	30.000		
Branching probability	0.125		
I/O 2 mean service (ms)	30.000		
Branching probability	0.125		
I/O 3 mean service (ms)	15.000		
Branching probability	0.250		
I/O 4 mean service (ms)	8.000		
Branching probability	0.500		

We ran six synthetic job mixes on a CDC 6400 configuration for further validation of the overlapped processing models. The configuration consisted of a CDC 6400 mainframe with 65,536 60-bit-words of main memory, seven peripheral processors, and two CDC 808 disk drives with associated controllers and channels. The job mixes were designed to fit entirely into main memory, allowing us to model the system as a simple two-I/O central server network with a constant level of multiprogramming. The I/O's had FCFS disciplines and the CPU had a round robin discipline with a fixed quantum of 16 ms. We approximated the CPU scheduler as FCFS in our analysis.

We ran three job mixes for each model, composed of 1, 2, or 4 identical synthetic programs. The level of

Table III. Job Mix Parameters.

CPU: I/O

Level of Multiprogramming	CPU Mean 1 (ms)	CPU Mean 2 (ms)	CPU Probability of Overlap	I/O 1 Branching Probability	I/O 1 Mean (ms)	I/O 2 Branching Probability	I/O 2 Mean (ms)
1	8.0	26.6	1.00	1.0	46.1	0.0	—
2	8.0	26.2	1.00	0.5	42.0	0.5	43.5
4	8.0	25.7	1.00	0.5	43.9	0.5	37.6

I/O: I/O

Level of Multiprogramming	CPU Mean 1 (ms)	CPU Mean 2 (ms)	CPU Probability of Overlap	I/O 1 Branching Probability	I/O 1 Mean (ms)	I/O 2 Branching Probability	I/O 2 Mean (ms)
1	78.6	9.0	1.00	0.5	35.2	0.5	35.8
2	67.5	9.0	1.00	0.5	39.9	0.5	39.6
4	67.6	9.0	1.00	0.5	38.0	0.5	45.8

multiprogramming 4 was chosen as being typical. The levels 1 and 2 were chosen to show maximum effects of overlap. The programs were written in Fortran and the CDC assembly language Compass using a local I/O software package, IOP [10]. IOP allows the user the option of nonoverlapped I/O or CPU:I/O overlap. We wrote two synthetic programs, one for each model, with the probability of overlap set to 1. The CPU:I/O overlapped program was written so the time for CPU_2 was approximately exponential, and so that CPU_1 represented the time necessary for IOP to set up the actual I/O transfer (a constant 8 ms). The I/O:I/O overlapped program was written so that CPU_1 was nearly exponential, and so that CPU_2 was the time to set up a data transfer (a constant 9 ms). In both programs, I/O consisted of transferring 512 60-bit words either from or to the disk. Each I/O request consisted of a seek, rotational delay, and the actual data transfer. The seek time and rotational delay were each uniformly distributed and the transfer time was constant. The I/O:I/O overlap program was written so that the two I/O requests were to separate disks. The nonoverlapped models were parameterized in the same way as for the simulation validations. The exact parameter values for the models were obtained from event trace data and are found in Table III.

The results from the overlap and nonoverlap models with parameters from Table III are found in Table IV. The results from the overlap model are remarkably good considering the various service distribution and service discipline approximations. In most cases, the tolerance for each performance measure was 0.05 for $p \geq 0.90$. The CPU mean wait time predictions for I/O:I/O overlap were within a tolerance of 0.10. The nonoverlap model predicted a performance measure more accurately than the overlap model in only one case: the CPU mean wait time was the *only parameter predicted more accurately* for

Table IV. Comparison of trace and model statistics.

Level of Multiprogramming	Approximation Model	Trace Measurements	Nonoverlapped Model
	CPU:I/O Model		
	CPU Util.		
1	0.57	0.57	0.45
2	0.87	0.91	0.75
4	0.97	1.00	0.92
	Mean CPU Queue Length		
1	0.57	0.57	0.45
2	1.43	*	1.06
4	3.14	3.18	2.60
	Mean CPU Wait Time		
1	34.6	34.6	34.6
2	57.0	*	51.6
4	109.0	108.5	104.6
	I/O:I/O Model		
	CPU Util.		
1	0.67	0.72	0.50
2	0.85	0.89	0.68
4	0.96	0.99	0.86
	Mean CPU Queue Length		
1	0.67	0.72	0.50
2	1.31	1.25	1.08
4	2.93	2.72	2.42
	Mean CPU Wait Time		
1	087.6	087.6	087.6
2	117.8	107.8	120.2
4	234.0	211.0	218.0

* Unobtainable due to garbage on tape.

I/O:I/O overlap with a level of multiprogramming of 4. In all the other cases, the nonoverlap model predicted performance measures much less accurately than the overlap model. For example, the utilizations predicted by the nonoverlap model ranged from 7 to 22 percent lower than the measured ones.

The evaluation algorithms were coded as Fortran routines and run on a CDC 6600. The CPU:I/O overlap models executed in less than 20 ms. and the I/O:I/O overlap models, for levels of multiprogramming less than 8, in less than 200 ms.

6. Comparison of Central Server Networks With and Without CPU:I/O Overlap

We consider a central server system with four I/O devices and where all servers have FCFS discipline. The I/O devices are identical and are selected with equal probability. We compare two systems identical in all respects, except that one has overlapping and the other does not. We are interested in the percentage improvement in CPU utilization which CPU:I/O overlapped processing provides. We want to look at the effects on this improvement that varying the level of multiprogramming N, the probability of overlap p, and relative CPU and I/O processing speeds may have. We define $\rho = \lambda/\mu$ as the parameter denoting the relative CPU and I/O processing speeds. Here $1/\mu$ is the mean CPU time between two I/O requests and $1/\lambda$ is the mean I/O time.

Figure 11 presents the improvement in CPU utilization for $p = 1.0$ and levels of multiprogramming 2 through 5 as the parameter ρ is varied. Here all computation takes place in CPU_2. Improvement is greatest when the system is well balanced (neither CPU nor I/O bound). The magnitude of improvement is sensitive to the level of multiprogramming and is negligible for high levels ($N > 4$). Figure 12 presents the improvement for a lower level of overlap, $p = 2/3$. For this case, improvement is also greatest when the system is well balanced and has a low level of multiprogramming ($N < 3$). For all values of N and ρ the improvement is significantly lower than in the case of full overlap. Figure 13 presents the improvement for levels of multiprogramming 2 through 5 as the parameter p is varied. For this case $\rho = 0.3$. For all levels of multiprogramming, the improvement curves are nearly linear functions of p. As the level of multiprogramming goes down, the improvement goes up and the curve becomes more sensitive to changes in p.

These curves may be readily explained. Overlapping is helpful only when it allows a device to be utilized which would not be utilized without overlapping. Consider a state in a nonoverlapped case where there is a job getting CPU service but no jobs waiting for or receiving service from an I/O device, say I/O device i. If overlapping were allowed in this state, then it is possible that the job receiving CPU service could be simultaneously processed on I/O device i, thus achieving higher overall processing rates. Now consider a state in a nonoverlapped case where there is a job receiving CPU service and a different job receiving service from I/O i. If overlapping between the CPU and I/O were permitted, it would not improve overall processing rates in this state. As the level of multiprogramming increases, the

Fig. 11. Percentage improvement in CPU utilization due to CPU: I/O overlap for $p = 1.0$.

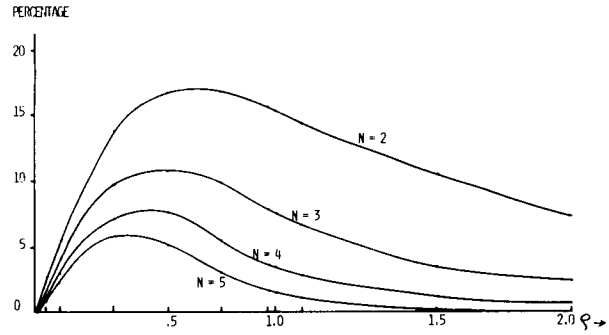


Fig. 12. Percentage increase in CPU utilization due to CPU: I/O overlap for $p = 2/3$.

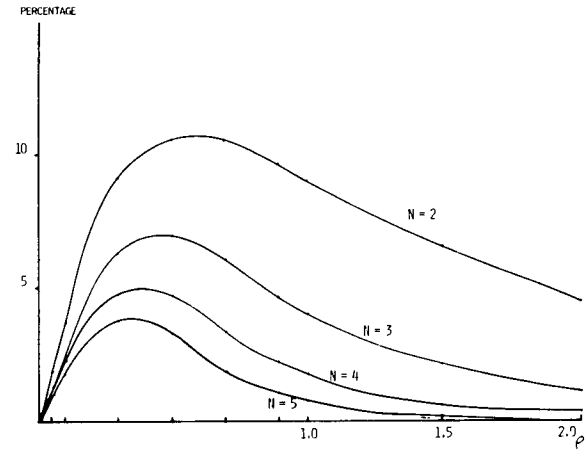
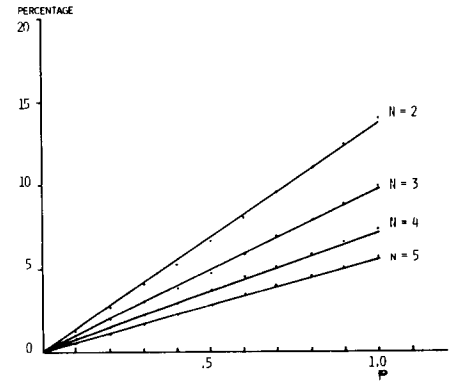


Fig. 13. Percentage increase in CPU utilization due to CPU: I/O overlap for $p = 0.3$.



fraction of time that the system spends in states where overlapping is advantageous decreases. Hence increased multiprogramming levels result in lower percentage improvement due to overlapping.

Now consider what happens when the system becomes increasingly I/O bound, and in particular consider the limiting case where the mean CPU service time is reduced to zero. In this case, the central server model is equivalent to one with a shorted CPU (Figure 3) and the thruput of this system is controlled by the I/O's. Allowing CPU:I/O overlap in the model of Figure 3 will not improve performance. A similar argument can be given to show that overlap will not improve performance if the system is CPU bound. Thus, the curves obtained from analysis mesh with our intuitive expectations.

The above figures indicate that only in special cases might overlapping CPU and I/O activities greatly increase performance. A system would have to be well balanced and with a low level of multiprogramming, possibly due to central memory limitations.

7. Comparison of Systems With and Without I/O:I/O Overlap

We consider a central server system with six I/O devices and all FCFS disciplines. The I/O devices have exponentially distributed service times with identical means and are selected with equal probability. We compare two systems which are identical in all respects except that one system has overlapping while the other does not. We are interested in the improvement which occurs by overlapping two I/O requests. We also want to study the effects of varying the level of multiprogramming N, the probability of overlap p, and the relative CPU and I/O speeds ρ. We make the assumption that no computation occurs during CPU_2, so that overlap benefits accrue exclusively from I/O:I/O overlap.

Figure 14 presents the relative improvement in CPU utilization for $p = 1.0$ over a range of levels of multiprogramming and values of ρ. The relative improvement tends to be greater for low values of ρ (when the system is I/O-bound) and for lower values of the level of multiprogramming. Figure 15 presents relative improvement curves for $p = 2/3$. The relative improvement is considerably less than for $p = 1.0$ which suggests that the value of p is critical. Figure 16 presents the relative improvement in CPU utilization for $\rho = 0.3$ as p is varied. As p increases, the relative improvement becomes more sensitive to changes in p. This is in marked contrast to the CPU:I/O overlap example. For high values of p, the level of multiprogramming is a less significant factor, unlike the CPU:I/O overlap example.

This above example indicates that high level of overlap is required ($p > 0.6$) before overlapping substantially affects CPU utilization. Overlapping of I/O requests is effective for systems in which jobs spend most of their time in I/O and in which the individual I/O devices are

Fig. 14. Percentage improvement in CPU utilization for system with I/O: I/O overlap, $p = 1.0$.

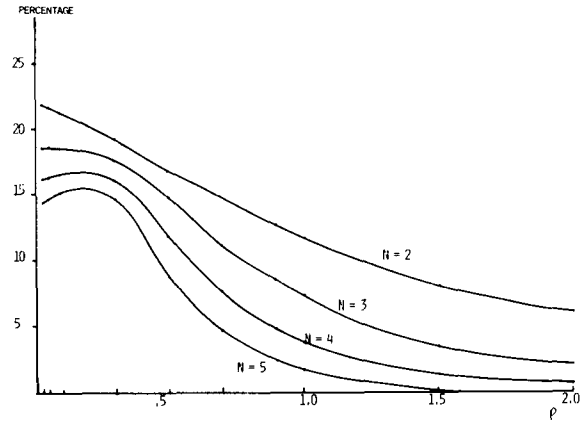


Fig. 15. Percentage improvement in CPU utilization for system with I/O: I/O processing, $p = 2/3$.

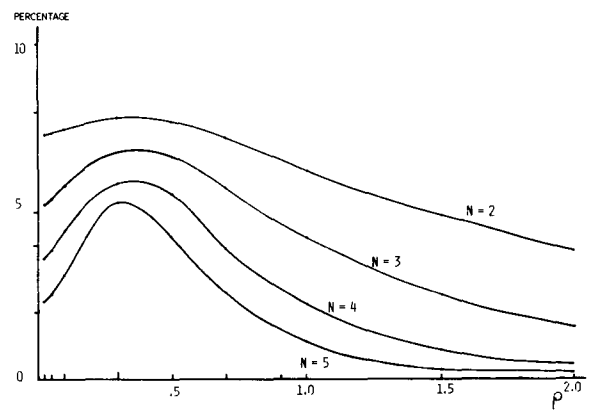
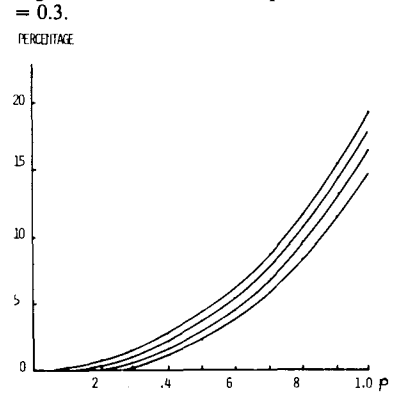


Fig. 16. CPU utilization improvement using CPU: I/O overlap for $p = 0.3$.



relatively underutilized. Database systems could possibly fall in the above category. In this case, as in the CPU:I/O overlap case, increasing levels of multiprogramming result in reduced percentage improvement due to overlap.

The curves for I/O:I/O overlap may also be readily explained. An I/O-bound system ($\rho \sim 0$) can be modeled

(approximately) by a shorted CPU, central server model (Figure 3). I/O:I/O overlap allows more I/O devices to be simultaneously utilized in this shorted CPU system thus increasing thruput. The percentage improvement in the $\rho = 0$ case depends upon the utilizations of the I/O devices which in turn depends upon the degree of multiprogramming and the number of I/O devices: if the degree of multiprogramming is high compared to the number of devices, then I/O device utilizations will be high and improvements due to I/O:I/O overlap will be small (for the same reasons as for the CPU:I/O overlap case). Note that for the $\rho = 0$ case (Figure 3), I/O:I/O overlap results in positive and usually substantial improvement in thruput whereas CPU:I/O overlap results in no improvement in thruput.

A CPU-bound system ($\rho \rightarrow \infty$) can be modeled approximately by a central server model with I/O's shorted out. In this case, I/O:I/O overlap does not improve thruput since thruput is completely controlled by the CPU service rate. Note that the percentage improvement deriving from overlapping decreases with ρ for low degrees of multiprogramming ($N = 2$), whereas it first increases and then decreases for high degrees of multiprogramming ($N = 5$). Since there are six disks, it follows in the $N = 2$ case that even if all the jobs spent all their time in the I/O subsystem, and even if the I/O's were overlapped, the disks would still not be saturated. Thus, the more time jobs spend in the I/O subsystem (i.e. the smaller the value of ρ), the greater the increase in I/O utilization derivable from overlapping. In the $N = 5$ case, if all the jobs spent all their time in the I/O subsystem, the disks would be heavily utilized even without overlapping. Overlapping cannot increase the already high utilization in this case. However, as ρ increases from a value of zero (0), the amount of time that jobs spend in the I/O subsystem decreases, and hence disk utilization decreases, in the nonoverlapped case, allowing for a significant increase in utilization from overlapping. Once again, analytic results agree with intuitive expectations.

8. Conclusions

We have presented solution techniques for the analysis of central server systems in which programs may exhibit CPU:I/O or I/O:I/O overlap. These techniques are compatible with other techniques used to include the effects of memory [3], passive resources [13, 14], nonexponential service time distributions [21], and multiple CPU's [22]; all these techniques use two-stage hierarchic models based on the use of Norton's Theorem followed by recursive analysis of Markov models of two-queue systems. These models were validated by running synthetic job mixes on a CDC 6400.

We used these models to consider the improvements obtained by overlapping CPU:I/O or I/O:I/O activities. CPU:I/O overlap appears to be of limited benefit, primarily because appreciable improvement occurs only for low levels of multiprogramming and high levels of overlap ($p \approx 1$). The benefits of overlapping several I/O requests are greater and less sensitive to the level of multiprogramming. However, high levels of overlap are still necessary. A critical parameter appears to be the number of I/O devices and their relative utilization. In either case, the designer of a system can resort to the models to determine whether the benefits will outweigh the costs.

It is also worth noting that the functional dependencies between "percentage improvement in thruput deriving from parallelism" and key system parameters are similar in the CPU:I/O overlap and I/O:I/O overlap cases treated here and the CPU:CPU overlap cases treated elsewhere [4, 23].

Received January 1977; revised January 1978

References

1. Baskett, F. Mathematical models of multiprogrammed computer systems. TSN-17, Comput. Ctr., U. of Texas at Austin, 1971.
2. Baskett, F., Chandy, K.M., Muntz, R.R., and Palacios-Gomez, F. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM 22*, 2 (April 1975), 248–260.
3. Brown, R.M., Browne, J.C., and Chandy, K.M. Memory management and response time. *Comm. ACM 20*, 3 (March 1977), 153–165.
4. Browne, J.C., Chandy, K.M., Hogarth, J. and Lee, C. The effect on throughput in multi-processing in a multi-programming environment. *IEEE Trans. Comptrs. 22* (1973), 728.
5. Buzen, J. Queueing network models of multiprogramming. Ph.D. Th., Div. Eng. and Appl. Physics, Harvard U., Cambridge, Mass., June 1971.
6. Chandy, K.M. The analysis and solutions for general queueing networks. Proc. Sixth Annual Princeton Conf. on Inform. Sci., Princeton U., Princeton, N.J., 1972, pp. 224–228.
7. Chandy, K.M., Herzog, U., and Woo, L. Parametric analysis of queueing network models. *IBM J. Res. and Develop. 19*, 1 (Jan. 1975), 36–42.
8. Chandy, K.M., Herzog, U., and Woo, L. Approximate analysis of general queueing networks. *IBM J. Res. and Develop. 19*, 1 (Jan. 1975), 43–49.
9. Chandy, K.M., Howard, J.H., and Towsley, D. Product form and local balance in queueing networks. *J. ACM 24*, 2 (April 1977), 250–263.
10. Computation Center. User's Manual. Computat. Ctr., U. of Texas at Austin, 1974.
11. Control Data Corp. A Simulation Process-Oriented Language (ASPOL) Reference Manual. Publ. 17314200, Control Data Corp., Minneapolis, Minn., 1972.
12. Herzog, U., Woo, L., and Chandy, K.M. Solution of queueing problems by a recursive technique. *IBM J. Res. and Develop. 19*, 3 (May 1975), 295–300.
13. Keller, T.W. Models of Computer Systems with Passive Resources. Ph.D. Th., Dept. Comptr. Sci., U. of Texas at Austin, 1976.
14. Keller, T.W. Central server models with peripheral processors. In *Computer Performance*, M. Reiser and K.M. Chandy, Eds., North-Holland Pub. Co., Amsterdam, 1977.
15. Lavenberg, S.S. Efficient estimation via simulation of workrates in closed queueing networks. IBM Res. Rep. RJ1390, IBM T. J. Watson Res. Ctr., Yorktown Heights, N.Y., 1974.
16. Mood, A. M., and Graybill, F.A. *Introduction to the Theory of Statistics.* McGraw-Hill, New York, 1963.
17. Maekawa, M., and Boyd, D.L. Two Models of Task Overlap with Jobs of Multiprocessing Multiprogramming Systems. Proc. 1976 Int. Conf. on Parallel Processing, Detroit, Aug. 1976, pp. 83–91.

18. Peterson, M., and Bulgren, W. Studies in Markov models of computer systems. Proc. 1975 ACM Annual Conf., Minneapolis, Minn., pp. 102–107.
19. Price, T.G. Models of multiprogrammed computer systems with I/O buffering. Proc. Fourth Texas Conf. on Comptng. Syst., U. of Texas at Austin, 1975.
20. Rose, C.A. Validation of a queueing model with classes of customers. Proc. Int. Symp. on Comptr. Performance Modeling, Measurement and Evaluation, Harvard U., Cambridge, Mass., March 1976, 318–325.
21. Sauer, C.H., and Chandy, K.M. Approximate analysis of central server models. *IBM J. Res. and Develop. 19*, 1, (Jan. 1975), pp. 301–313.
22. Sauer, C.H., and Chandy, K.M. Parametric modeling of multi-miniprocessor systems. IBM Res. Rep. RC5978, IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., 1976.
23. Shedler, G.S. A cyclic queue model of a paging machine. IBM Res. Rep. RC2814, IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., 1970.
24. Smith, W.L. Renewal theory and its ramifications. *J. Royal Statist. Soc. B20* (1958), 243–302.
25. Towsley, D. Local Balance Models of Computer Systems. Ph.D. Th., Dept. Comptr. Sci., U. of Texas at Austin, 1975.

Programming Techniques

S.L. Graham, R.L. Rivest
Editors

Jump Searching: A Fast Sequential Search Technique

Ben Shneiderman
University of Maryland

When sequential file structures must be used and binary searching is not feasible, jump searching becomes an appealing alternative. This paper explores variants of the classic jump searching scheme where the optimum jump size is the square root of the number of records. Multiple level and variable size jump strategies are explored, appropriate applications are discussed and performance is evaluated.

Key Words and Phrases: jump searching, sequential files, file management, search strategies, database structures, index searching

CR Categories: 3.74, 4.34

1. Introduction

Locating a record with a given target key, or determining its absence from a file, is a central problem in file management. For sequentially organized files which are sorted by a single key field, Knuth [4] describes a variety of useful algorithms. If storage has been allocated contiguously, binary searching can be used to produce very good performance. Interpolation or curve fitting techniques based on linear or higher degree polynomials [10] have seen limited use because of the high cost of computation and uneven performance. Sequential searching is simple to program but is costly, when compared to binary searching. If records are placed in a random access memory and linked together with explicit pointers, the binary tree search technique or its variants [7] are preferred.

Author's address: Dept. of Information Systems Management, University of Maryland, College Park, MD 20742.