# PRINCIPLES OF OPTIMALITY FOR MULTIPROGRAMMING

Jacques LEROUDIER and Dominique POTIER

IRIA - LABORIA
BP 5.
78150 LE CHESNAY
FRANCE

## Summary

In recent papers [1,2] we have presented an analytical model of virtual memory system performance which was used to provide a characterization of thrashing and some insight into the behaviour of virtual memory computer systems. We pointed out the existence of an optimal degree of multiprogramming with respect to the central processing unit (CPU) utilization and it was shown that the value of the optimal degree of multiprogramming was closely related to such program behaviour parameters as locality and I/O rate. These observations were used to support a method to control the degree of multiprogramming in a virtual memory computer system by estimating dynamically the activity of the resources of the system [2].

In this paper, we shall use the same analytical model to investigate the behaviour of the paging drum when the degree of multiprogramming is set to its optimal value. The main conclusion, which corroborates the general feeling on performance of virtual memory systems [3,14, 5], is that drum utilization remains in the 50 °/o range whenever CPU utilization is maximized, if no resource is saturated. The results are validated through simulation experiments in order to relax some theoretical assumptions used in the analytical model and to take into account some detailed mechanisms such as CPU overheads. As an application, an adaptive control algorithm based on the activity of the paging device is proposed. The algorithm has been simulated and comparisons with results reported in [2] are presented.

Moreover, after investigations we show also that optimal performance can only be achieved through a balanced use of the different resources of the system. This conclusion cross-checks and extends Buzen's theoretical results [6] and Wulf's practical ones [7].

## The Model

The model represented in figure 1 has been described in detail in [1,2]. It consists of a CPU, a secondary memory (SM) device and a filing disk (FD). A queue of requests is associated with each device and the order in which these requests are satisfied is assumed to be FIFO.

A fixed number n of statistically identical and independent processes (user programs) execute in the system, so that if at any time, $n_0$, $n_1$, $n_2$ are the numbers of processes in the CPU, SM, FD queues, respectively, then

$$n = n_0 + n_1 + n_2 \qquad (1)$$

The behaviour of processes is characterized by a computation time followed by either a page fault after which the process enters the SM queue or an I/O request in which case the process enters the FD queue. Processes which terminate their service at the SM of at the FD return to the CPU queue. The feedback loop around the CPU queue represents processes which depart from the system but which are immediately replaced by a new process, thus maintaining a constant degree of multiprogramming.

Let $T_{sm}$, $T_{fd}$ be the mean service times of the SM and the FD, respectively and $T_{pf}$ and $T_{io}$ the virtual mean times between page faults and I/O operations.

From the work-rates theorems due to Chang and Lavenberg [8] we can derive the following relations between $T_{sm}$, $T_{fd}$, $T_{pf}$, $T_{io}$ and the CPU, SM and FD utilizations, respectively $A_{cpu}(n)$, $A_{sm}(n)$ and $A_{fd}(n)$:

$$A_{sm}(n) = \frac{T_{sm}}{T_{pf}} A_{cpu}(n) , \qquad (2)$$

$$A_{fd}(n) = \frac{T_{fd}}{T_{io}} A_{cpu}(n) . \qquad (3)$$

It should be noted that these equations can be written without any assumptions about the distribution functions involved in the model. If we assume exponential distributions, the model is then a particular case of Jackson's network [9] and the stationary solution of the system $p(n_0, n_1, n_2)$ is known and given by

$$p(n_0, n_1, n_2) = \frac{1}{G(n)} \left( \frac{T_{sm}}{T_{pf}} \right)^{n_1} \left( \frac{T_{fd}}{T_{io}} \right)^{n_2} , \qquad (4)$$

where

$$G(n) = \sum_{n_1, n_2} \left( \frac{T_{sm}}{T_{pf}} \right)^{n_1} \left( \frac{T_{fd}}{T_{io}} \right)^{n_2} . \qquad (5)$$

In [10] Buzen showed that the CPU utilization can be written

$$A_{cpu}(n) = \frac{G(n-1)}{G(n)} \qquad (6)$$

and proposed an algorithm to compute $G(n)$, and hence $A_{cpu}(n)$, which will be used to obtain the numerical results presented in the next sections of the paper.

It remains to characterize $T_{pf}$ which depends on the process behaviour but also on the system and, in particular, on the memory allocation policy. Assuming that the memory is allocated to a process on a page-on-demand basis we shall use experimental evidence to relate the mean time $T_{pf}$ between two consecutive page faults of a program (process in the system) to the amount of space allocated to this program. Belady and Kuehner [11] have proposed the following model to fit their measurements :

$$T_{pf} = am^k \qquad (7)$$

A more complex model presented by Chamberlin and Liu [12] was also used. However, all the computations we have done show no significant differences between the results obtained from the two models, and we shall therefore restrict our attention in the sequel to the first life-time model.

Furthermore, we shall assume that the core memory available is of size M and that it is equally shared among processes (user programs) which yields :

$$m = \frac{M}{n} , \qquad (8)$$

$$T_{pf} = a \left(\frac{M}{n}\right)^k . \qquad (9)$$

To summarize, the model is characterized by the following set of parameters :

| | |
|---|---|
| $n$ | degree of multiprogramming, |
| $M$ | total memory size, |
| $T_{sm}$, $T_{fd}$ | mean service times at the SM and FD stations, |
| $a$, $k$ | parameters of the life-time model, |
| $T_{io}$ | virtual mean time between I/O operations, |

and we shall investigate in the next section the behaviour of $A_{cpu}(n)$, $A_{sm}(n)$, $A_{fd}(n)$ for a wide range of these parameters.

## The Criterion Under Consideration

We are interested by two kinds of performance measures. We want, on one hand, to have a good utilization of the different resources in the system and, on the other hand, to ensure a satisfactory system's response time. These two imperatives can be conciliated by considering the ratio of the real execution time of n programs with no multiprogramming to the real execution time of n programs with multiprogramming of degree n. $D(n)$, or what we call "dilatation", measures the expansion of the real time which is obtained by multiprogramming n programs. We can evaluate $D(n)$ if we notice that $D(n)$ is also the ratio of the program throughputs with and without multiprogramming. Let $T_c$ be the mean total computation time of a program and let $T_{pf}(1)$ be the mean time between page faults when a program is alone in the system. Then, the program throughput with no multiprogramming is :

$$\left[T_c + \frac{T_c}{T_{io}} T_{fd} + \frac{T_c}{T_{pf}(1)} T_{sm}\right]^{-1} ,$$

and when the degree of multiprogramming is n, the program throughput is simply :

$$\frac{A_{cpu}(n)}{T_c} .$$

Hence, D(n) is expressed by :

$$D(n) = A_{cpu}(n)(1 + \frac{T_{fd}}{T_{io}} + \frac{T_{sm}}{T_{pf}(1)}) , \qquad (10)$$

where $T_{pf}(1) = a(\frac{M}{1})^k$ .

From (2) and (3), we can write :

$$D(n) = A_{cpu}(n)(1 + \frac{T_{sm}}{T_{pf}(1)}) + A_{fd}(n) \qquad (11)$$

So D(n) is not only a measure of the system response time but also of the resource utilizations, since it can be expressed as their sum. The term $\frac{T_{sm}}{T_{pf}(1)}$ comes from the existence of a virtual memory, and it can usually be neglected, if the size of the memory is appropriate because then $T_{pf}(1)$ is large compared to $T_{sm}$. Hence D(n) can be approximated by :

$$D(n) \simeq A_{cpu}(n) + A_{fd}(n). \qquad (12)$$

It should be noticed that equation (12) is similar to the one written in [2, eq.(7)] which was obtained with another approach. In [2] "dilatation" is defined as a measure of the parallelism between the resources of the system. Thus, our criterion takes into account the three following measures :

- parallelism between system resources,
- resource utilizations,
- system's response time,

which are shown to be related.

## Secondary Memory Utilization

Having defined the model and a measure of performance we can now use the model to investigate the behaviour of the secondary memory when the degree of multiprogramming is set to the value which maximizes D(n). From equation (10), it can be noticed that D(n) is also maximized when $A_{cpu}(n)$ is maximized and therefore we shall restrict our attention in this section to $A_{cpu}(n)$.

Three series of experiments have been performed and the results are presented in the figures 2, 4 and 5. The experiments consist in computing $A_{cpu}(n)$ and $A_{sm}(n)$ for different values of three important parameters of the model : the mean service time $T_{sm}$ of the secondary memory, the mean time between I/O requests $T_{io}$, the coefficient k of the life-time function. The functions $A_{cpu}(n)$ and $A_{sm}(n)$ are then plotted on the same graph so that the behaviour of the secondary memory can be simply related to the measure of performance $A_{cpu}(n)$, and hence to D(n).

Several observations can be made from the figures 2, 4 and 5. First of all, all the $A_{sm}(n)$ curves are monotonic increasing S shaped functions, with an inflexion point located around $A_{sm}(n) = 0.5$ which roughly corresponds to the maximum value of $A_{cpu}(n)$. In order to check this point the portions of $A_{cpu}(n)$ and $A_{sm}(n)$ corresponding to the values of n for which $0.4 \leq A_{sm} \leq 0.6$ have been outlined. We can observe that for all the experiments the outlined portions correspond to the maximum values of $A_{cpu}(n)$, and it can be consequently inferred that the optimum working range of the system can be simply characterized by the load of the secondary memory device. The second observation is important with respect to this first conclusion : the fact that the inflexion point of $A_{sm}(n)$ is located within the optimum working range means sensitivity $\frac{\Delta n}{\Delta A_{sm}(n)}$ is minimum in this zone. The practical implication of this result is that an optimum degree of multiprogramming can be satisfactorily obtained from the estimation of $A_{sm}(n)$.

The last observation to be made is that these conclusions have been obtained for a wide range of architecture and process behaviour parameters, $T_{sm}$, $T_{io}$, k, and it can therefore be thought that they do not depend on a particular combination of parameters, but express some basic aspects of the behaviour of multiprogrammed virtual memory computer systems. It would have been more satisfactory to derive our results from some mathematical properties of $A_{cpu}(n)$ and $A_{sm}(n)$ rather than from more observations of the curves. Unfortunately, we have been unable to exhibit such properties, and, to our knowledge, this is still an open problem. To end with, the model has been extended and simulated in order to take into account CPU overhead and constant service times at the SM. The results are presented in figure 3 and point out the same relation between $A_{cpu}(n)$, the user's useful computation, and $A_{sm}(n)$.

It remains to verify that the results we have obtained are in good agreement with experimental observations made on multiprogrammed virtual memory systems. The general opinion shared by people dealing with such systems is that optimum performance is achieved when there is practically no request waiting behind the paging channel which means that the secondary memory load does not exceed 0.6 or so. For instance, Rodriguez-Rosell states in [3] that "measurements of the paging and disk file channel queues indicate that before trashing no queues are formed at the I/O channels". During discussions with Adams [5] about the EMAS system it was pointed out that their system was operating at its maximum efficiency when the paging drum channel was busy fifty percent of the time. More precise figures can be derived from measurements made on the IBM-CP67 system [3, fig. 1 and 3] and performance prediction made on the MULTICS system [14, table 5-7, p 227]. It should be noted that in the IBM-CP67 case, the fraction of memory pages which has been written is missing in [3] and was taken to be equal to 0.70 from a discussion with J. Rodriguez-Rosell. The results are presented in Tables I and II and show good agreement with the observations derived from the model.

The results which have been obtained should not be viewed as a mathematical property of the system under study but rather as an indication that in such a system there exists a strong relation between its optimum working point and the load of the secondary memory. The derivation of mathematical model was useful to point out the behaviour of the system for different parameters and to exhibit the particular shape of the functions $A_{cpu}(n)$ and $A_{sm}(n)$. To summarize, the conclusion we have arrived at can be worded into the following principles.

<u>Principle I</u>

In a multiprogrammed virtual memory computer, optimum performance is achieved if the utilization of the secondary memory remains in the 50 °/₀ region.

<u>Principle II</u>

At the optimum working point the sensitivity of the degree of multiprogramming to the utilization of the secondary memory is minimum.

These two principles have direct implications for the regulation of the degree of multiprogramming. Principle I gives a simple indicator of optimum system behaviour. Principle II says that an optimum degree of multiprogramming can be simply estimated by measuring the utilization of the secondary memory and that errors in the measurements will not a fect much the estimation of the optimum degree of multiprogramming. In the next section, we present an adaptive algorithm for the control of the degree of multiprogramming based on these two principles.

## Adaptive Control of the Degree of Multiprogramming

The function of an adaptive control is to maintain the system in operation so that the measure of performance D(n) is optimized. The control scheme is based on the two Principles stated in the previous section and the structure of the controller represented in figure 6 is kept similar to the one presented in [2] : an estimator EST of system behaviour uses measurements on the secondary memory device to estimate its utilization ; an optimizer OPT computes decisions using the estimates provided by EST as well as a knowledge of the system state ; OPT furnishes commands to the switch K controlling the access of users waiting in the queue Q into the resource loop. The solid lines in figure 6 indicate the cycle a user will follow and the dotted lines represent the flow of information to the various units of the controller.

The controller has been implemented in a simulation model of a virtual memory time-sharing system. This simulation model is an extension of the analytical model described in section II where CPU overheads and non-exponential service time distributions are also taken into account. The different units of the controller perform the following operations.

<u>1) EST</u> : Let $X_i$ be the most recent value of the secondary memory utilization which is measured.

EST computes a smoothed unbiaised estimate $Z_i$ of the secondary memory utilization as follows [2]. Let $Y_i$ be given by

$$Y_i = \alpha X_i + (1-\alpha)\, Y_i-1, \qquad 0 < \alpha < 1,$$

then

$$Z_i = \lambda_i\, Y_i$$

where

$$\lambda_i = \frac{1}{1 - (1-\alpha)^i}$$

The variance $V(Z_i)$ is computed as well in order to build a confidence interval $[z^+, z^-]$ of the secondary memory utilization.

<u>2) OPT</u> : Updates the current estimate n* of the optimal degree of multiprogramming according to the estimate of the secondary memory utilization provided by EST. OPT is based on a threshold S, which is a parameter of the controller chosen in the 0.5 region from Principle I. OPT, when it is activated, computes an optimal degree of multiprogramming n* which is transferred to K :

if $S \in [z^+, z^-]$ n* remains n*
if $S > [z^+]$ n* is increased by 1.
if $S < [z^-]$ n* is decreased by 1.

It should be recalled that the control scheme takes into account only the introduction of processes into the multiprogramming set, and that processes cannot be forced to leave the system.

In the simulations which have been run, S has been chosen equal to 0.5. However, in a real system some tuning up of the controller would certainly be necessary in order to determine the best value of S.

The simulation experiments presented in [2] have been exactly repeated. They consist in running a single simulation with two different loads: LOAD II for 200 sec., then LOAD I for 300 sec., then LOAD II for 200 sec. LOAD I and LOAD II have different localities and I/O rates so that the optimum degree of multiprogramming is 6 for LOAD I and 18 for LOAD II. The same improvements in system performance as in [2] have been obtained and, moreover, it can be shown by comparing the figures 7a and 7b that the control based on the secondary memory utilization adapts itself to load variations three times faster than the previous scheme based on CPU and FD utilizations. One reason for this result is that in the present case the decision is based on the estimation of one point : the secondary memory utilization, rather than on the estimation of a function as in [2]. It should also be noted that the measurements and the operations performed by EST and OPT are simpler in the present case.

## Balancing CPU and FD

We have presented in the previous section an adaptive control scheme which estimates the size of the multiprogramming set which maximizes system performance and ensures that thrashing is prevented. However, another level of optimization is still necessary in order to determine, within the multiprogramming set, the mix of programs which will ensure optimal utilization of both CPU and FD.

This problem of improving system performance by balancing the load of the different resources is commonly felt as a crucial one by computer center managers and has been studied by several authors [4, 6, 7, 15]. However, most of these studies, excepting [7], concern the partition of a given load among different peripheral units and do not investigate the question of balancing CPU and I/O devices in relation with paging activity.

In order to examine this problem we use the analytical model presented in section II and the measure of performance D(n) defined in section III.

Let $D^* = D(n^*)$ be the value of the measure of performance for the optimal degree of multiprogramming n*. In other words, D* represents the system performance when the size of the multiprogramming set is optimum. We can now look at the behaviour of D* as a function of $\frac{T_{io}}{T_{fd}}$ for different values of k in order to study the performance of the optimum multiprogramming set size for different values of the mean I/O rate of the programs belonging to this set. D* is plotted in figure 8 as a function of $\frac{T_{io}}{T_{fd}}$ for k varying from 1.5 to 2.5. It can be observed that D* is maximized for values of

$\frac{T_{io}}{T_{fd}}$ close to 1, and that the sensitivity of D* to $\frac{T_{io}}{T_{fd}}$ is largely dependant on k. For large values of k, i.e. $k \geq 2$, overall performance is maximized for $\frac{T_{io}}{T_{fd}} = 1$, which means the utilizations of CPU and FD are balanced as can be seen from equation (3). When k is small, i.e. $k < 2$, the optimum is obtained for $\frac{T_{io}}{T_{pf}} < 1$. However, in this case, there is no sensible variation of D* with $\frac{T_{io}}{T_{fd}}$ due to the fact that paging then dominates system activity.

From these observations, we can state the following principle :

## Principle III

An optimal mix of programs within the multiprogramming set is achieved when the mean virtual time between consecutive I/O of the programs belonging to the multiprogramming set is equal to the mean service time of the I/O device.

Principle III provides the basis of a control scheme for selecting which program enters the multiprogramming set : the program should be selected so that the quantity $T_{io}$ measured for the multiprogramming set is as close as possible to the mean service time $T_{fd}$ of the I/O device.

It can be noticed that Principle III rejoins the intuitive control based on balanced device utilizations proposed by Wulf in [7]. Another obvious consequence of Principle III is that fastest devices should be more loaded than slower ones. This point has been analyzed in detail by Buzen [6] and similar conclusions were obtained.

## Conclusion

We have presented in this paper a two level optimization of multiprogrammed virtual memory computer performance. The basic principles of the optimization schemes have been derived from the analysis of a model of virtual memory time-sharing system and experimental observations have been presented to verify our conclusions. The first level of optimization, based on the utilization of the secondary memory, determines the optimal size of the multiprogramming set, and the second level of optization balances the utilization of the CPU and of the I/O devices. Simulation experiments of the first level have shown a significant improvement with regards to [2]. The implementation of the second level of optimization is presently being done. It sets some new problems about the exponential smoothing of the estimates which have been pointed out by Wulf [7] and requires further investigations in order to determine the values of the smoothing constants.

## References

1. M. Badel, E. Gelenbe, J. Lenfant, J. Leroudier, D. Potier - "Adaptive optimization of the performance of virtual memory computer" - Proceedings of the workshop on computer architecture and networks - IRIA - August 1974.

2. M. Badel, E. Gelenbe, J. Leroudier, D. Potier - "Adaptive optimization of a time-sharing system's performance" - IEEE Proceedings on Interactive Computer System - June 1975.

3. J. Rodriguez-Rosell, J.P. Dupuy - "The evaluation of a time-sharing page demand system" - Spring Joint Computer Conference 1972.

4. P.J. Denning - "Multiprogrammed memory management" - IEEE Proceedings on Interactive Computer Systems - June 1975.

5. J.C. Adams, G.E. Millard - "Performance measurements on the Edinburgh Multi Access System" - ICS 75 - Antibes - June 1975.

6. J.P. Buzen - "Analysis of system bottlenecks using a queueing network model" - ACM-SIGOPS - Workshop on System performance evaluation - Harvard - April 1971.

7. W.A. Wulf - "Performance monitors for multiprogramming systems" - Second Symposium on Operating Systems Principles - Princeton - October 1969.

8. A. Chang, S.S. Lavenberg - "Work-rates in closed queueing networks with general independent servers" - IBM Research Report RJ 989 - 1972.

9. J.R. Jackson - "Jobshop like queueing systems"- Management Science 10,, 1 - October 1963.

10. J.P. Buzen - "Computational algorithm for closed queueing networks with exponential servers"- CACM, Vol. 16, n° 9 - September 1973.

11. L.A. Belady, C.J. Kuehner - "Dynamic space-sharing in computer systems" - CACM, Vol. 12, n° 5 - May 1969.

12. D.D. Chamberlin, S.H. Fuller, L.Y. Liu - "A page allocation strategy for multiprogramming systems" - IBM Research Report RC 3848 - May 1972.

13. J.C. Adams - "Architecture and performance evaluation of EMAS" - Seminars on modeling and system performance evaluation - IRIA - October 1975.

14. A. Sekino - "Performance evaluation of multiprogrammed time-shared computer systems" - Ph.D. Thesis - Project MAC TR-103 - MIT - September 1972.

15. C.G. Moore - "Network models for large scale time-sharing systems" - Ph.D. Thesis - University of Michigan 1971.

| n | 1 | 2 | $\boxed{3}$ | .4 | 5 | 6 |
|---|---|---|---|---|---|---|
| system efficiency [3] | 0.17 | 0.28 | 0.31 | 0.29 | 0.21 | 0.15 |
| Real page fault rate [3] (sec.$^{-1}$) | 9 | 16 | 24 | 30 | 36 | 42 |
| Secondary memory utilization | .18 | .33 | .49 | .61 | .73 | .85 |

TABLE I

Secondary memory utilization in IBM - CP67.

(The length of time required to complete a page fetch is taken equal to 1.70 the mean service time of the secondary memory to take into account the swap out of written pages, which yields $T_{sm}$ =1.70 × 12 =20.4 msec).

| n | 1 | 2 | $\boxed{3}$ | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| user's useful computation [14] $A_{cpu}(n)$ | 55.4 | 63.7 | 65.5 | 65.1 | 63.9 | 62.5 | 61.3 | 60.2 |
| mean virtual time between page fault [14] $T_{pf}(n)$ (msec.) | 98.5 | 55.9 | 41.7 | 34.7 | 30.5 | 27.7 | 25.8 | 24.3 |
| page fault rate $A_{cpu}(n)/T_{pf}(n)$ (msec.) | .562 | 1.14 | 1.57 | 1.88 | 2.10 | 2.26 | 2.36 | 2.48 |
| secondary memory utilization $A_{cpu}(n)\ \frac{T_{pf}(n)}{T_{sm}}$ | 0.20 | 0.40 | $\boxed{0.55}$ | 0.66 | .73 | .79 | .83 | .87 |

TABLE II

Secondary memory utilization in MULTICS

($T_{sm}$ = 35 msec.)

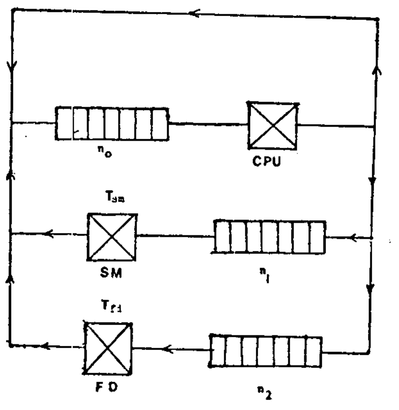

FIGURE 1

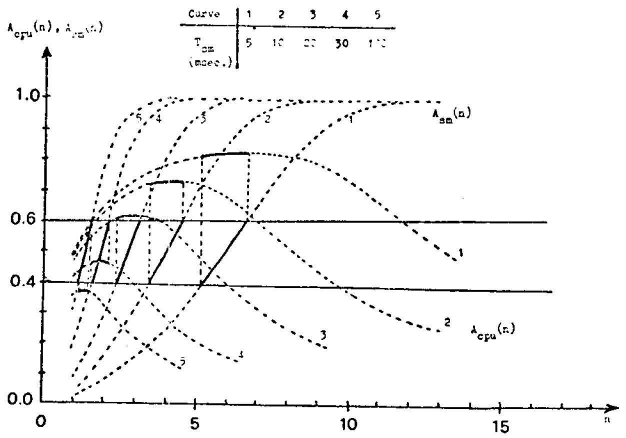

Figure 2 : $A_{cpu}(n)$ and $A_{sm}(n)$ as functions of n

M = 512, k = 1.5, a = 0.01msec., $T_{io}$ = 30 msec., $T_{fd}$ = 30 msec.

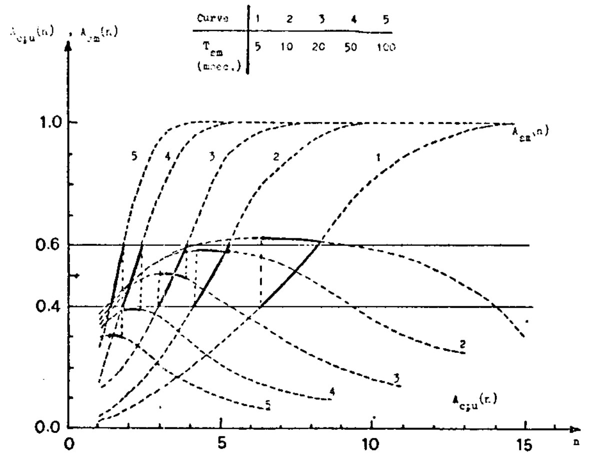

Figure 3 : $A_{cpu}(n)$ and $A_{sm}(n)$ as function of n

. M = 512, k = 1.5, a = 0.01 msec., $T_{io}$ = 30 msec., $T_{fd}$ = 30 msec.

| | |
|---|---|
| Context switching overhead | 0.5 msec. |
| file i/O overhead | 2.0 msec. |
| paging i/O overhead | 1.0 msec. |

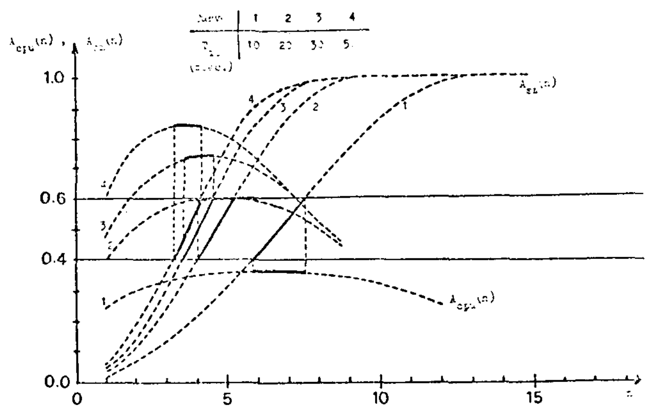

Figure 4 : $A_{cpu}(n)$ and $A_{sm}(n)$ as functions of n

M = 512, k = 1.5, a = 0.01 msec., $T_{fd}$ = 30 msec., $T_{sm}$ = 10 msec.

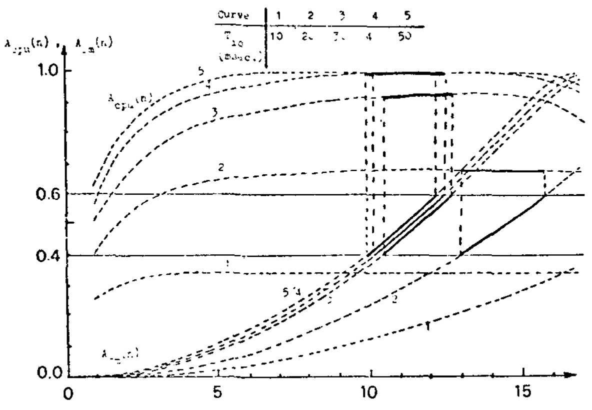

Figure 5 : $A_{cpu}(n)$, $A_{sm}(n)$ as functions of n

M = 512, k = 2, a = 0.01 msec., $T_{fd}$ = 30 msec., $T_{sm}$ = 10 msec.

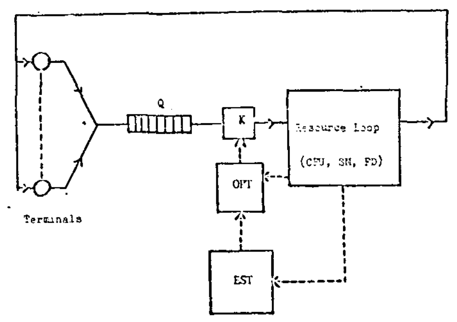

Figure 6

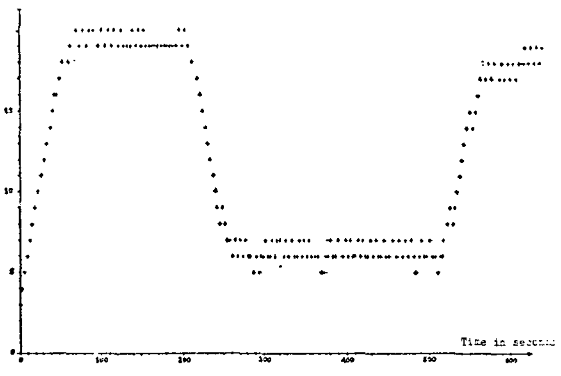

Figure 7a : Estimate of the optimal degree of multiprogramming obtained by applying Principles I and II.

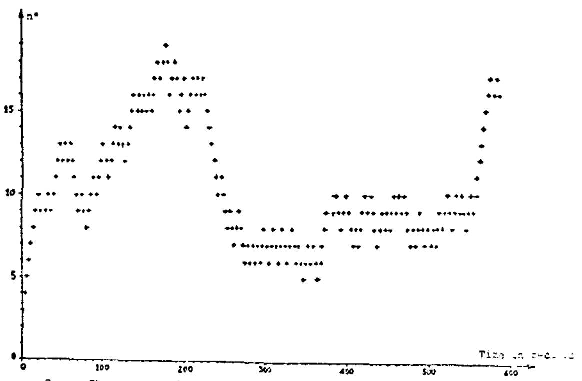

Figure 7b : Estimate of the optimal degree of multiprogramming obtained from experiments reported in [2].

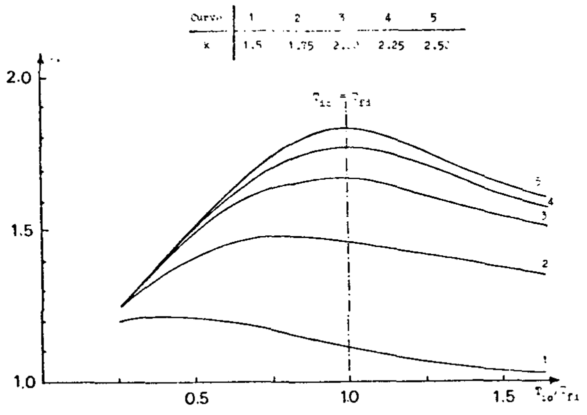

Figure 8 : n* as function of $T_{io}/T_{fd}$

M = 25[illegible], a = [illegible].1, $T_{sm}$ = 10 msec.