OPTIMAL MULTIPROGRAMMING :
PRINCIPLES AND IMPLEMENTATION

Marc BADEL

Jacques LEROUDIER

IRIA - LABORIA
B.P. 105
78150 Le Chesnay
FRANCE

ABSTRACT

Three principles of optimality for multiprogramming are derived from a general model of a virtual memory computer system. They state the existence of both an optimal multiprogramming degree and an optimal program mixture in the multiprogramming set. The implementation of these principles is carefully studied in a simulation model which permits, in contrast to analytical models, to relax assumptions on the workload submitted to the system. More precisely the workload is supposed to be non-stationary and issued from a non-homogeneous program population. Therefore a dynamic control of the system needs special estimators. These statistical problems are studied in detail and an implementation is described. The results obtained on the system performance improvement are presented.

INTRODUCTION

In previous papers [1, 2] we described a general model of a virtual memory computer system and studied it both from a static viewpoint by predicting the performance in the case of stationary assumptions and from a dynamic one by controlling the system in the case of a non-stationary workload. More specifically in [3, 4], we derived from our model some simple rules for optimizing a virtual memory system. The present paper is dealing with the implementation of these rules and it emphasizes the underlying assumptions and the various statistical techniques needed for such an implementation.

In the first section we briefly recall the model and the performance criterion we consider as well as the three principles of optimality for multiprogramming we pointed out. In the second section we describe the implementation of the first two principles and more particularly the statistical difficulties we had to overcome. The third section is devoted to the third principle and the specific statistical problem encountered. In the last section we present the experiments we have conducted and the results obtained. Finally we consider several performance

criteria and we show they can be satisfied simultaneously.

It is worth noting that the statistical results we derive solve general problems occurring when estimating mean values of non-stationary stochastic processes. Therefore they can be widely used out of the context in which we apply them.

1 - THE MODEL AND THE PERFORMANCE CRITERION

1.1 - The model

The model represented in figure 1 has been described in detail in [1,2].

Figure 1

It consists of a CPU, a secondary memory (SM) device (e.g., fixed head disk or drum), a file disk (FD) and a set of terminals. A queue of requests is associated with each device and the order in which these requests are satisfied is assumed to be FIFO. There is no queue in front of the terminals since each terminal is dedicated to a user (i.e. a

user's program or a process).

If N is the number of terminals n_0 , n_1 , n_2 the numbers of processes, respectively, in the CPU queue, SM queue and FD queue, the degree of multiprogramming n is defined by

$$(1) \quad n = n_0 + n_1 + n_2, \quad 0 \leq n \leq N$$

The degree of multiprogramming represents the number of processes (or programs) sharing the memory.

The processes behaviour is characterized by a computation time followed either by a page fault after which the process enters the SM queue or by an I/O request in which case the process enters the FD queue. Processes which terminate their service at the SM or at the FD return to the CPU queue.

Obviously, the degree of multiprogramming n does not remain constant, it varies according to the arrivals of processes from terminals and their departures from the multiprogramming set (because a program has completed its computation time). Therefore it is not possible to derive a steady state solution of the model (values of figures of merit such as queue lengths, resource utilizations, etc...) for each value of n . However we will compute such a solution by assuming our model is near completely decomposable [5, 6], that is there is time enough between two variations of n (two terminal interactions) to ensure we reach stationarity. The main underlying assumption is concerned with terminal interactions which are supposed to be rare compared with internal interactions in the system. In this case, Courtois [5, 6] shows that the figures of merit we are looking for can be derived for a given n by computing them at steady state for this value of n .

In order to verify this assumption, we simulated our model and estimated our figures of merit according to two strategies :

- first by running n experiments, one for each value of n ,
- secondly by running only one experiment, in which n can fluctuate according to the arrivals into or the departures from the system.

We found out identical results (confidence intervals overlapped) by these two approaches, thus we considered our model was actually near-completely decomposable.

Let T_{sm} , T_{fd} respectively be the mean service times of the SM and the FD, 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 [9], we can derive the following relations between T_{sm} , T_{fd} , T_{pf} , T_{io} and the CPU, SM and FD utilizations, respectively denoted by $A_{cpu}(n)$, $A_{sm}(n)$ and $A_{fd}(n)$:

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

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

It should be noted that these equations can be written without any assumption on the distribution functions involved in the model. If we assume exponential distributions, the model is then a particular case of Jackson's network [7, 1] and the stationary solution of the system $p(n_0, n_1, n_2)$ is known and given by

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

where

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

In [8] Buzen shows that the CPU utilization can be written as :

$$(4) \quad A_{cpu}(n) = \frac{G(n-1)}{G(n)}$$

and an algorithm is proposed for computing $G(n)$, and hence $A_{cpu}(n)$.

The relations (2) and (3) assume there is no overhead, that is no CPU time is consumed when such an event as a page fault or an I/O occurs. If we denote respectively by θ_{pf} , θ_{io} and θ_{cpu} the mean overheads when a page fault, an I/O or a program completion happens, and by T_{cpu} the mean virtual time of program executions, it is shown in [10] that :

$$(5) \quad A'_{cpu}(n) = A_{cpu}(n) \frac{1}{1 + \frac{\theta_{cpu}}{T_{cpu}} + \frac{\theta_{pf}}{T_{pf}} + \frac{\theta_{io}}{T_{io}}}$$

where $A_{cpu}(n)$ is the total CPU utilization and $A'_{cpu}(n)$ the only CPU utilization consumed by program users (without overhead).

Therefore the relations (2) and (3) become :

$$(6) \quad A_{sm}(n) = \frac{T_{sm}}{T_{pf}} A'_{cpu}(n)$$

$$(7) \quad A_{fd}(n) = \frac{T_{fd}}{T_{io}} A'_{cpu}(n)$$

It should be noted that since

$$A_{cpu}(n) \leq 1$$

we have :

$$(8) \quad A'_{cpu}(n) \leq \frac{1}{1 + \frac{\theta_{cpu}}{T_{cpu}} + \frac{\theta_{pf}}{T_{pf}} + \frac{\theta_{io}}{T_{io}}}$$

It remains to characterize T_{pf} which depends on the program behaviour but also on the system and, in particular, on the memory allocation policy. Assuming that the memory is allocated to a program 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 to the amount of space allocated to this program. Belady and Kuehner [12] have proposed the following model to fit their measurements :

$$(9) \quad T_{pf} = am^k$$

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

$$m = \frac{M}{n}$$

and

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

We studied in [13, 14] (see also [1]) the different values of a and k for several programs. We pointed out that these two parameters are not independent but related to each other by :

$$(10) \quad k \approx -\log a + 1 \text{ or } a \approx 10^{-k+1}$$

1.2 - The criterion under consideration

We are interested in two kinds of performance measures. We want, on the one hand, to have a good utilization of the different system resources 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 a multiprogramming degree n . $D(n)$, also called "dilatation" [2, 3, 10], measures the expansion of the real time which is obtained by multiprogramming n programs. We can evaluate $D(n)$ by noting that $D(n)$ is also the ratio of the program throughputs with and without multiprogramming (see [10]). Then, $D(n)$ is expressed by :

$$(11) \quad D(n) = A'_{\text{cpu}}(n) \left(1 + \frac{T_{\text{fd}}}{T_{\text{io}}} + \frac{T_{\text{sm}}}{T_{\text{pf}}(1)} + \frac{\theta_{\text{pf}}}{T_{\text{pf}}} + \frac{\theta_{\text{pf}}}{T_{\text{pf}}(1)} + \frac{\theta_{\text{io}}}{T_{\text{io}}} \right)$$

where $T_{\text{pf}}(1) = a \left(\frac{M}{I} \right)^k$

From (3), we can write :

$$(12) \quad D(n) = A'_{\text{cpu}}(n) + A_{\text{fd}}(n) + A'_{\text{cpu}}(n) \left(\frac{T_{\text{sm}}}{T_{\text{pf}}(1)} + \frac{\theta_{\text{pf}}}{T_{\text{pf}}(1)} + \frac{\theta_{\text{pf}}}{T_{\text{pf}}} + \frac{\theta_{\text{io}}}{T_{\text{io}}} \right)$$

So $D(n)$ is not only a measure of the system response time, but also a measure of the resource utilizations, since it can be expressed as their sum. The term $\frac{T_{\text{sm}}}{T_{\text{pf}}(1)}$ comes from the existence of a virtual

memory. It can usually be neglected, if the size of the memory is appropriate because $T_{\text{pf}}(1)$ is large compared to T_{sm} . The other terms

$\frac{\theta_{\text{pf}}}{T_{\text{pf}}(1)}$, $\frac{\theta_{\text{pf}}}{T_{\text{pf}}}$ and $\frac{\theta_{\text{io}}}{T_{\text{io}}}$ come from the overheads : in general, they can be neglected too.

To summarize, if the system is well sized, the third term in (12) is negligible (if not, it is not worthwhile of optimizing anything), hence $D(n)$ can be approximated by :

$$(13) \quad D(n) \approx A'_{\text{cpu}}(n) + A_{\text{fd}}(n)$$

It should be noticed that equation (13) 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.

1.3 - Three principles of optimality

In [3] (see also [4]), we pointed out three principles for optimizing the "dilatation" $D(n)$.

Principle I :

In a multiprogrammed virtual memory computer, optimum performance is achieved if the utilization of the secondary memory (SM) remains in the 50% region.

Principle II :

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

Principle III :

An optimal mixture of programs within the multiprogramming set is achieved when the mean virtual time between consecutive I/O requests on a specific device of program belonging to the multiprogramming set is equal to the mean service time of the I/O device.

We also indicated a possible implementation of the first two principles. In the next section, we rapidly recall this implementation and the underlying assumptions, but we will insist upon the statistical problems we have encountered.

It was important to recall the properties of our model, because the proof of the relations (2), (3), (5), (6), (7) and (12), without any distributional assumptions, can only be found in [10], and these relations are essential for implementing the third principle (see §3).

2 - IMPLEMENTATION OF THE 50% RULE

The control of the degree of multiprogramming we have implemented is based on the first two principles. The structure of the whole model

is represented in figure 2.

Figure 2

An estimator EST of system behaviour uses measurements on the secondary memory device to estimate its utilization. An optimizer OPT computes decisions by using the estimates provided by EST as well as the 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.

This control is described in detail in [3, 11]. It mainly consists of building a confidence interval for the SM utilization in order to decide the admission or the rejection of programs. Because of our assumption of non-stationarity we have to use special estimators which are able to forget the past. We first presented such estimators in [1, 2]; they are said to be exponentially smoothed. We recall that we estimate a mean value by :

$$(14) \quad \hat{E}(X) = Y_i = \alpha X_i + (1-\alpha)Y_{i-1}, \quad 0 < \alpha \leq 1$$

(X_i being the i -th measurement of the variable X)

where the "smoothing" coefficient α can be chosen so as to determine the relative importance of "old" measurements.

In order to estimate $V(\hat{E}(X))$ we are led to estimate

$$(15) \quad V(X_i) = E(X_i^2) - (E(X_i))^2$$

Because of the non-stationarity, we have to use smoothed estimators again. In [1, 2, 3] we proposed to consider a "smoothing" coefficient β for estimating $E(X^2)$:

$$(16) \quad \hat{E}(X_i^2) = U_i = \beta X_i^2 + (1-\beta)U_{i-1}, \quad 0 < \beta \leq 1$$

The problem is to ensure that the two estimates ($\hat{E}(X_i)$ smoothed with α and $\hat{E}(X_i^2)$ with β) age at the same "speed". If such a precaution is not taken, the expression (15) can be negative when the stochastic process $X(t)$ is varying along the time. It is this point we discuss in this section, for we encountered some difficulties with it. We refer the reader to [2, 3, 11] for the other details of the 50% rule implementation.

A given α being selected for stationarity reasons, we are searching a β in order to get the expected aging effect.

It is clear that the relation between α and β depends on the stochastic process $X(t)$ (here, the SM utilization) and more particularly on its first two moments. This explains that, while we found a satisfactory solution with $\beta = \alpha$ [1, 2] when $X(t)$ was the dilatation (a sum of utilizations), this is not any more the case when $X(t)$ is the SM utilization.

From (14) we have :

$$(17) \quad V(Y_i) = \alpha^2 V(X_i) + (1-\alpha)^2 V(Y_{i-1})$$

hence

$$V(Y_i) = \alpha^2 V(X) (1 + (1-\alpha)^2 + \dots + (1-\alpha)^{2(i-1)})$$

finally

$$(18) \quad V(Y_i) = \alpha V(X) \frac{1 - (1-\alpha)^{2i}}{2 - \alpha}$$

Thus

$$(19) \quad \lim_{i \rightarrow \infty} V(Y_i) = \frac{\alpha V(X)}{2 - \alpha}$$

Now we will study the behaviour of our estimators on a sudden variation of $E(X)$ and $V(X)$ in order to determine an order of magnitude of β with respect to α .

Assume this variation occurs at time $t=0$ and we know two unbiased estimators E_0 and V_0 of $E(X)$ and $V(X)$ at this time. Assume too that the measurement indexing starts at $t=0$.

From (19), it comes

$$(20) \quad V(E_0) = \frac{\alpha V_0}{2 - \alpha}$$

and from (14), (16), (17) :

$$(21) \quad E(Y_i) = \alpha E(X_i) + (1-\alpha) E(Y_{i-1})$$

$$(22) \quad V(Y_i) = \alpha^2 V(X_i) + (1-\alpha)^2 V(Y_{i-1})$$

$$(23) \quad E(U_i) = \beta E((X_i)^2) + (1-\beta) E(U_{i-1})$$

with the relation :

$$(E(Y_i))^2 = E(Y_i^2) - V(Y_i)$$

If the system has been initiated a sufficiently long time ago, we may consider there is no bias due to its starting point, and equation (25) in [2] gives :

$$(24) \quad E[\widehat{V}(Y_i)] = \frac{\alpha}{1 - \alpha} (E(U_i) - E[(Y_i)^2]) + \frac{1 - \alpha}{1 + \alpha} E[\widehat{V}(Y_{i-1})]$$

Let B_i be the bias (due to the variations of E_0 and V_0) on the variance of Y_i after i measurements. By definition, we have :

$$(25) \quad B_i = E[\hat{V}(Y_i)] - V(Y_i)$$

Owing to the recurrence equations (21), (22), (23) and (24), we can compute B_i as a function of i and β only (with α given), if we know the initial conditions (at the time $t=0$) :

$$E(Y_0) = E(E_0)$$

$$E(U_0) = V(E_0) + E(E_0^2) = \frac{\alpha}{2-\alpha} V_0 + E(E_0^2)$$

$$V(Y_0) = V_0$$

$$E(\hat{V}(Y_0)) = V_0$$

For each β , we can compute :

$$\max_i |B_i|$$

where i ranges from 1 to L , L being large enough so that B_i remains constant (B_i converges). We can take the smallest of these different values obtained for different values of β :

$$(26) \quad \min_{\beta} (\max_i |B_i|)$$

Let β_0 be the value of β for which the minimum (26) is reached. Then β_0 is the value of β we are looking for, since it ensures that the bias introduced by the sudden variation at time $t=0$ is minimal.

By this way, we numerically determined convenient values for β . From our experiments, it seems that :

$$\beta = 0.8\alpha$$

is an acceptable value for α lying in the interval :

$$0.05 \leq \alpha \leq 0.20$$

and for a relatively large variation of the stochastic process $X(t)$ (since $X(t)$ is the SM utilization) :

$$\text{mean variation : } 0.33 \rightarrow 0.95$$

$$\text{variance variation : } 4 \times 10^{-4} \rightarrow 13 \times 10^{-4}$$

It is this value of β we have used to implement the 50% rule.

We have been bent on presenting these specific results because they appeared to be important even if we neglected them in a first approach

[1, 2].

We do not recall the optimization we obtain in applying the 50% rule to control the degree of multiprogramming (see [2, 3, 11]). In the sequel of the paper we assume this optimization has been achieved and we are trying to optimize the program mixture of the multiprogramming set by applying our third principle.

3 - IMPLEMENTATION OF THE THIRD PRINCIPLE

3.1 - Balancing the resource utilizations

Let us recall that the criterion we optimize reads (cf. (12) and (13)) :

$$D(n) \approx A'_{\text{cpu}}(n) + A_{\text{fd}}(n)$$

or (cf. (7))

$$(27) \quad D(n) \approx A'_{\text{cpu}}(n) \left(1 + \frac{T_{\text{fd}}}{T_{\text{io}}}\right)$$

With the 50% rule we are able to exhibit a n which maximizes (27), that is :

$$(28) \quad D^* = D(n_0) = A'_{\text{cpu}}(n_0) \left(1 + \frac{T_{\text{fd}}}{T_{\text{io}}}\right) \geq D(n) \quad \forall n$$

Up to now we have always assumed the homogeneity of the program population, if this does not hold, there exist sub-populations with specific program behaviours. That means, in terms of our model, there exist programs with different T_{io} (mean virtual time between I/O requests).

Our third principle is stating that D^* (see (28)) is optimal if :

$$(29) \quad T_{\text{io}} = T_{\text{fd}} \Leftrightarrow \frac{T_{\text{fd}}}{T_{\text{io}}} = 1$$

that is, from (7), all the resource utilizations (except the SM utilization which is kept in the vicinity of 50%) are equal.

T_{fd} is an architecture parameter and it will be considered fixed. But T_{io} is a program behaviour parameter and, if there exist program sub-populations (it is mostly the case in the reality), by a convenient program mixture in the multiprogramming set we can create a resulting T_{io} such that it is equal to T_{fd} . It is the implementation of this program mixture control we describe in this section.

To validate this last optimization, we will extend the model studied by considering several I/O devices, formally m devices and three in our simulation model (see section 4).

The extended model is represented in figures 3 and 4.

Figure 3

Figure 4

This model consists of :

- 1 - a program SOURCE modelling both interactive arrivals from a set of terminals and batch arrivals.
- 2 - a resource SET composed of one CPU, one paging unit and m I/O devices (see figure 4).
- 3 - a POOL of programs split into a FIFO queue for programs just ente-

ring the system and a BAG for the others.

- 4 - a module EST in charge of picking up measurements on the system state and the behaviour of each program submitted to the system (EST is also in charge of updating the different statistical estimates).
- 5 - a module OPT which defines an optimal degree of multiprogramming and a system "state".
- 6 - a module K which both decides the admission of a program into or its rejection from the SET and chooses this program.

To summarize , the global control we proposed is based upon

- the control of the degree of multiprogramming by applying the 50% rule.
- the control of the program mixture by applying our third principle.

Notice that if we consider m I/O devices, the third principle reads :

$$(30) \quad T_{io_j} = T_{fd_j} \quad \forall j, 1 \leq j \leq m$$

3.2 - The implementation

From (30) we deduce :

$$(31) \quad \frac{T_{fd_j}}{T_{io_j}} = 1 \quad \forall j, 1 \leq j \leq m$$

where T_{fd_j} are constant architecture parameters.

At a first glance, comparing $\frac{T_{fd_j}}{T_{io_j}}$ to 1 seems to be equivalent to

comparing $\frac{T_{io_j}}{T_{fd_j}}$ to 1. For statistical reasons it is not true.

T_{fd} is a perfectly known constant, so it is quite similar to have it in the numerator or in the denominator. On the other hand T_{io_j} is an estimate which can vary from small values - very frequent I/O, thus T_{io_j} is often measured and well known - to large one -rare I/O, rarely and badly known. Therefore when T_{io_j} is ill determined (thus very dispersed) its numerical value is very large and fluctuates heavily. In order

to avoid this annoying effect we are led to consider the quantity

$\frac{1}{T_{ioj}}$ which is small when fluctuating, and therefore to compare $\frac{T_{fdj}}{T_{ioj}}$ to

one.

Now the problem is to decide, at a given instant, what program we have to

- inject into the system (n_0 increases, $A_{sm}(n)$ is under 50%)
- reject from the system (n_0 decreases, $A_{sm}(n)$ is above 50%)
- exchange in the system (n_0 remains constant, $A_{sm}(n)$ is equal to 50%)

in order to respect the relation (31).

To perform the comparisons we use an absolute value norm which is relatively simple and cheap to implement.

The $\frac{T_{fdj}}{T_{ioj}}$ estimates resulting from the system state (the mixture in the

multiprogramming set) are not directly evaluated. They are derived from the Chang-Lavenberg equations (see (7)) :

$$(32) \quad \frac{T_{fdj}}{T_{ioj}} = \frac{A_{fdj}(n)}{A'_{cpu}(n)} \quad \forall j, 1 \leq j \leq m$$

that is from resource utilization estimates which we previously talked about (see (14), (24) and [2, 3, 11]).

For the moment we will assume we are able to estimate (see section 3.3)

quantities such as $\frac{T_{fdj}}{T_{ioj,k}}$ where $T_{ioj,k}$ stands for the mean virtual

time between I/O request of the program k for the I/O unit j.

If n is the current degree of multiprogramming and m the number of I/O devices in the system, we propose the following scheduling algorithm :

① - Candidate to enter the system (condition $n < n_0$)

It is the program in the POOL for which the following lower bound is reached :

$$(33) \quad \min_{k \in \text{POOL}} \sum_{j=1}^m \left| n \frac{A_{fd,j}(n)}{A'_{cpu}(n)} + \frac{T_{fd,j}}{T_{io,j,k}} - (n+1) \right|$$

② - Candidate to leave the system (condition $n > n_0$)

It is the program in the main memory (in the SET) for which the following lower bound is reached :

$$(34) \quad \min_{k \in \text{SET}} \sum_{j=1}^m \left| n \frac{A_{fd,j}(n)}{A'_{cpu}(n)} - \frac{T_{fd,j}}{T_{io,j,k}} - (n-1) \right|$$

③ - Candidate to be exchanged (condition $n = n_0$)

First, a program is rejected into the POOL (more precisely into the BAG) by applying ② and then a new program is injected into the SET by applying ① (it can be the same as the one rejected).

It is noteworthy that quantities such as $\frac{T_{fd,j}}{T_{io,j,k}} = \frac{A_{fd,j}(n)}{A'_{cpu}(n)}$ may vary

from 0 to $+\infty$, but that they are bounded (lying in the interval $]0, 1]$) when the resource j is underloaded ($A_{fd,j}(n) < A'_{cpu}(n)$) and otherwise possibly infinite (see figure 5).

Figure 5

Therefore in (33) and (34) overloaded resources have heavier weights than underloaded ones. This effect favours the underloading of a resource. So to avoid that, we propose to weight a resource according to its utilization.

Our weighting (see figure 6) is equal to 1 when the resource is overloaded and linearly increasing when the underloading appears. Note that this weighting must also be increasing with the number m of resources in the system.

Figure 6

Thus we propose the weight q_j for each resource j :

$$(35) \quad q_j = \begin{cases} 1 & \text{if } \frac{T_{fd_j}}{T_{io_{j,k}}} \geq 1 \\ (1-ma_j) \frac{T_{fd_j}}{T_{io_{j,k}}} + ma_j & \text{otherwise} \end{cases}$$

where a_j is a coefficient related to physical characteristics of the resource j . Then, expressions (33) and (34) become :

$$(36) \quad \min_{k \in \text{POOL}} \sum_{j=1}^m q_j \left| n \frac{A_{fd_j}(n)}{A'_{cpu}(n)} + \frac{T_{fd_j}}{T_{io_{j,k}}} - (n+1) \right|$$

$$(37) \quad \min_{k \in \text{SET}} \sum_{j=1}^m q_j \left| n \frac{A_{fd_j}(n)}{A'_{cpu}(n)} - \frac{T_{fd_j}}{T_{io_{j,k}}} - (n-1) \right|$$

The last problem we are faced before tackling the statistical one is dealing with the measurement initialization.

When a program arrives in the system from the SOURCE it is queued in the FIFO queue in the POOL of programs. It must be noted that we have no information on it. Two policies are possible. On the one hand we may guess we know some "a priori" information on programs and use it to initialize measurements, on the other hand we may front the hard reality and admit we do not know anything about program behaviour. It is this last hypothesis we assume.

In this context, we propose to initialize the $T_{ioj,k}$'s measurements of each program k with the T_{fd} 's value. That is we assume that an unknown program makes a "balanced" (in the sense of our third principle) use of the system resources. The aim of such an initialization is to favour the entrance of new programs only when the system is balanced and therefore when our criterion is optimized (cf. principle III).

3.3 - Estimation of the $T_{ioj,k}$'s

The control of the program mixture we propose needs to estimate utilizations such as $A_{fdj}(n)$ and $A'_{cpu}(n)$, and quantities such as the $T_{ioj,k}$'s or the $\frac{1}{T_{ioj,k}}$'s (see § 3.2). Estimating utilizations has already been solved in previous papers [2, 3, 11] (see also § 2). We now are solving the estimation problem of the $\frac{1}{T_{ioj,k}}$'s.

On the contrary to the utilization case, the $\frac{1}{T_{ioj,k}}$'s cannot be evaluated at anytime since they need the occurrence of an event, here an I/O request. However we would like to dispose of $\frac{1}{T_{ioj,k}}$'s estimations at anytime, in particular, when a decision has to be taken.

In a first step, we will assume that the virtual time intervals between the I/O requests of a program k for a resource j are exponentially distributed. Let us consider the Bernoulli's variable $X(t)$ defined by

$$X(t) = \begin{cases} X(t) = 1 & \text{if an event has occurred between the times } t \text{ and } t + \Delta t \\ X(t) = 0 & \text{otherwise} \end{cases}$$

where Δt is a time interval arbitrary small.

To be able "to forget the past", we propose to estimate the $\frac{1}{T_{ioj,k}}$'s by :

$$(38) \quad \left[\widehat{\frac{1}{T_{ioj,k}}} \right] = L(t + \Delta t) = \alpha X(t) + (1 - \alpha \Delta t) L(t)$$

where α is a smoothing coefficient.

If (38) is unbiased, we must have :

$$E(L(t)) = \frac{1}{T_{io_{j,k}}}$$

so let us compute $E(L(t))$. It comes from (38)

$$E(L(t + \Delta t)) = \alpha E(X(t)) + (1 - \alpha \Delta t) E(L(t))$$

From the exponential assumption we get

$$E(X(t)) = \frac{\Delta t}{T_{io_{j,k}}}$$

Hence

$$E(L(t + \Delta t)) = \alpha \frac{\Delta t}{T_{io_{j,k}}} + (1 - \alpha \Delta t) E(L(t))$$

When $\Delta t \rightarrow 0$ we get the differential equation :

$$\frac{d(E(L(t)))}{dt} = \frac{\alpha}{T_{io_{j,k}}} - E(L(t))$$

Therefore

$$(39) \quad E(L(t)) = \frac{1}{T_{io_{j,k}}} + C e^{-\alpha t}$$

Thus if C is not equal to zero, our estimation is biased but this bias is decreasing with t .

Notice that if $E(L(0)) = \frac{1}{T_{io_{j,k}}}$, then C is equal to zero, but in general

$\frac{1}{T_{io_{j,k}}}$ is unknown and the initialization which is performed is

$$E(L(0)) = 0$$

In this case $C = -\frac{1}{T_{io_{j,k}}}$.

Implementation of our estimator.

Let us consider a time interval $[t_\ell, t_m]$ in which no event has occurred, so (38) becomes :

$$L(t + \Delta t) = (1 - \alpha \Delta t) L(t)$$

by integrating we get :

$$(40) \quad L(t) = L(t_\ell) e^{-\alpha(t-t_\ell)} \quad \forall t \in [t_\ell, t_m]$$

where $L(t_\ell)$ is known (in the limit $L(t_\ell)$ is $L(0)$, a given initial value).

The algorithm we propose is as follows. Assume t_ℓ is the last instant such as $L(t_\ell)$ is known, two possible cases :

- ① - An event occurs at time t (necessarily the first one after t_ℓ), then

$$L \leftarrow L \times e^{-\alpha(t-t_\ell)} + \alpha$$

$$t_\ell \leftarrow t$$

- ② - No event at time t , but we want to estimate $L(t)$, then

$$L \leftarrow L \times e^{-\alpha(t-t_\ell)}$$

$$t_\ell \leftarrow t$$

Thus an estimation of $L(t)$ is available in L at any time. Remark that the estimate L is decreasing with t for a given t_ℓ . That means $L(t)$ gets smaller and smaller if no event occurs. This effect is in complete agreement with what we want.

Validation of our approach

We will now consider the estimator L defined by our algorithm (see ① and ②) and study it when no assumption is made concerning the distribution of the virtual time intervals between events.

The occurrence of an event (in our context, an I/O request) is a discontinuity point for $L(t)$ and $E(L(t))$, since just after an event $E(L(t))$ is changed into $E(L(t)) + \alpha$ (cf. ①). We will study $L(t)$ just after an event.

Let $t_1, t_2, \dots, t_n$ be the instants of event occurrences and t_0 , the starting instant. Let us consider the time intervals between events $\tau_1, \tau_2, \dots, \tau_n$, where

$$\tau_i = t_i - t_{i-1}$$

From the point ① of our algorithm, we derive

$$(41) \quad L(t_i) = L(t_{i-1}) e^{-\alpha\tau_i} + \alpha$$

So

$$(42) \quad E(L(t_i)) = E(L(t_{i-1})) e^{-\alpha\tau_i} + \alpha$$

If we assume that $L(t_{i-1})$ and $e^{-\alpha\tau_i}$ are independent, we get :

$$(43) \quad E(L(t_i)) = E(L(t_{i-1}))E(e^{-\alpha\tau_i}) + \alpha$$

For the moment, let us assume that the τ_i 's are i.i.d. (further we will relax this stationarity by tuning our forgetfulness parameter α) and f is their common density function. Then

$$\forall i \quad E(e^{-\alpha\tau_i}) = \int_0^\infty e^{-\alpha x} f(x) dx$$

If we note f^* the Laplace transform of f , we can write :

$$\forall i \quad E(e^{-\alpha\tau_i}) = f^*(\alpha)$$

Then the equation (43) becomes :

$$E(L(t_i)) = E(L(t_{i-1})) f^*(\alpha) + \alpha$$

Since $f^*(\alpha) < 1$, we can solve this recurrence equation :

$$(45) \quad \lim_{i \rightarrow \infty} E(L(t_i)) = \frac{\alpha}{1 - f^*(\alpha)}$$

Thus our estimator converges.

Notice that if the time intervals between events are exponentially distributed, we get :

$$(46) \quad \lim_{i \rightarrow \infty} E(L(t_i)) = \frac{1}{T_{io_{j,k}}} + \alpha$$

Therefore it is trivial to unbiased the estimator.

Now we will express $f^*(\alpha)$ without any assumption concerning f .

The parameter α is a smoothing parameter, so we will assume it is close enough to zero to develop $f^*(\alpha)$:

$$f^*(\alpha) = 1 - \alpha E(f) + \dots + (-1)^k \frac{\alpha^k}{k!} E(f^k) + o(\alpha^{k+1})$$

Hence it comes from (45):

$$\lim_{i \rightarrow \infty} E(L(t_i)) = \frac{\alpha}{1 - (1 - \alpha E(f) + \frac{\alpha^2}{2} E(f^2) + o(\alpha^3))}$$

Thus

$$(47) \quad \lim_{i \rightarrow \infty} E(L(t_i)) = \frac{1}{E(f)} + \frac{\alpha}{2} + \frac{\alpha}{2} \frac{V(f)}{(E(f))^2} + o(\alpha^2)$$

$E(f)$ is the mean virtual time between events, that is $T_{i0j,k}$. If we note K the variation coefficient of f ($K^2 = \frac{V(f)}{(E(f))^2}$), we obtain :

$$(48) \quad \lim_{i \rightarrow \infty} E(L(t_i)) = \frac{1}{T_{i0j,k}} + \frac{\alpha}{2} + \frac{K^2 \alpha}{2} + o(\alpha^2)$$

Therefore we deduce from this equation :

- 1 - our estimation is biased and in the Poisson case ($K=1$) the bias is α (see (46)).
- 2 - the bias is minimum ($\frac{\alpha}{2}$) when the time intervals are constant ($K=0$).
- 3 - in general, we can control the bias by choosing α . If $B_{\max}$ is the maximum bias we accept, from (48) we derive

$$\alpha \leq \frac{2B_{\max}}{1 + K^2}$$

- 4 - we have to find out a trade-off between :
 - the bias, a small α
 - the distribution stationarity, a sufficiently large α to forget the past fast enough.

L is the estimator we have used to estimate the $\frac{1}{T_{i0j,k}}$'s. To avoid a

too important overhead, we tabulated such expressions as $e^{\alpha(t_k - t_l)}$ when we implemented our control.

4. VALIDATING OUR CONTROL

4.1 Experiments on the mixture optimality

We built a simulation model as described in figures 3 and 4, and we implemented our control in it.

Having previously validated our control of the degree of multiprogramming (see [2, 3, 11]), in these experiments, we are mostly concerned with the control of the program mixture.

Of course the improvement obtained by a suitable program mixture greatly depends on the heterogeneity of the program population submitted to the system. Therefore to measure this improvement we considered

extreme conditions. Three kinds of workloads are generated by the SOURCE. All of them are similar concerning their memory (T_{pf}) and CPU (T_{cpu}) behaviours, but they differ on their I/O behaviour. The LOAD I uses the I/O device 1 ($T_{io_{1,k}} = 30 \text{ msec.}$) without using the other devices ($T_{io_{2,k}} = T_{io_{3,k}} = \infty$). The LOAD II and III are defined in the same manner, that is :

$$\text{LOAD II } T_{io_{1,k}} = T_{io_{3,k}} = \infty \text{ and } T_{io_{2,k}} = 30 \text{ msec.}$$

$$\text{LOAD III } T_{io_{1,k}} = T_{io_{2,k}} = \infty \text{ and } T_{io_{3,k}} = 30 \text{ msec.}$$

In other respects the I/O devices characteristics are identical ($T_{fd_1} = T_{fd_2} = T_{fd_3} = 100 \text{ msec.}$).

In such a context, it is clear that none of the three LOADS is a "good" workload for the system and clear too that we need to combine them.

We have defined an experiment in which the three loads are not naturally mixed together. The SOURCE sequentially and cyclicly generates programs according to the characteristics of the loads I, II and III. The mean time between each load is 4 secondes.

We performed this experiment twice. In the two cases, we controlled the degree of the multiprogramming with the 50% rule, but on the one hand the program mixture is not controlled and on the other hand it is. The result obtained are presented in figures 7 and 8.

We plot the instantaneous "dilatation", measured in the system, as a function of the time.

In the first experiment (without control) the loads are not naturally mixed together and since they are equivalent from the dilatation viewpoint ($T_{io_{j,k}}$'s symetry in the $D(n)$ expression), this criterion is uniformly low, fluctuating around 1.3 (hardly better than in monoprogramming).

In the second experiment, the optimization due to the program mixing is effective from the arrival of a program which is of a different "type". So, according to the arrivals of loads I, II and III, the dilatation is gradually increasing from 1.5 to 2.7 and to 3. Therefore we double

Figure 7

Figure 8

the system performance.

Two remarks are noteworthy. On the one hand the mixture control is near instantaneous, so we may conclude that our estimator L of the

$\frac{1}{T_{ioj,k}}$'s is quite relevant and the tuning of α is right. On the other

hand the criterion fluctuations increase with the mixture efficiency (injection and rejection of program coming from an heterogeneous population). Therefore, in such a context, it is utopian to control the system by looking for a criterion maximum (see [2]). We have to use an indirect optimization such as the one provided by the 50% rule (see also [4]).

4.2 - The "losers"

Up to now the proposed scheduling algorithm aims to optimize a unique performance criterion : the dilatation.

The dilatation enables to measure the usage of the different system resources. So its optimization will ensure a good use of all the system components. With an assumption of homogeneous program population, we showed in [2, 11] that the mean program response time is also optimized.

Since this assumption does not hold any longer, this means that programs can be "forgotten" if their behaviours do not fit the "state" of the system. To avoid this phenomenon detrimental not to the system performance but to the user's satisfaction, we must introduce a new criterion in the optimization in order to take into account this satisfaction.

In fact, we would like to consider an arbitrary number of criteria. As a matter of fact a computer system consists of several layers with several scheduling policies. The layer we consider is very close to the physical resources, but the scheduling algorithm we propose has to be able to take into account the constraints (economical costs, priorities, etc.) coming from other layers. It is this problem we attempt to solve in this section. In a first step we will restrict ourselves to the losers' problem. In this view we have to

- recognize a "forgotten" program
- to favour such a program by an appropriate mechanism.

In order to recognize a forgotten program we propose to compare its "efficiency" (see [2, 11]) with the system "efficiency". We recall that

program efficiency between the two instants t_{i-1} and t_i is the sum of the service times provided to the program during this time interval divided by $t_{i-1} - t_i$. We show in [2, 11] that the "dilatation" is the sum of the program efficiencies. So we derive the system efficiency as:

$$(49) \quad e_s = \frac{D(n)}{n}$$

To take into account the services provided to the programs in POOL, we generalize (49) by considering :

$$(50) \quad e_s = \frac{D(n)}{\ell}$$

where $\ell = n + \text{number of programs in POOL}$.

If e_k is the efficiency of program k , let us consider

$$(51) \quad s_k = \frac{e_k}{e_s}$$

Notice that the greater the coefficient s_k is, the better the program k has been served. So we propose to define a penalty for each program k such that :

$$(52) \quad p_k = C \frac{s_k}{\gamma_k}$$

where C and γ_k ($0 < \gamma_k \leq 1$) are fixed positive constants.

Thus the relations (36) and (37) become respectively :

$$(53) \quad \min_{k \in \text{POOL}} \left(\sum_{j=1}^m q_k \left| n \frac{A_{fd,j}(n)}{A_{cpu}(n)} + \frac{T_{fd,j}}{T_{io,j,k}} - (n+1) \right| + p_k \right)$$

$$(54) \quad \min_{k \in \text{SET}} \left(\sum_{j=1}^m q_k \left| n \frac{A_{fd,j}(n)}{A_{cpu}(n)} - \frac{T_{fd,j}}{T_{io,j,k}} - (n-1) \right| + p_k \right)$$

Therefore the proposed penalty penalizes programs better served than the system state permits and it favours the others.

The coefficient C enables to tune the penalties weights with regard to the dilatation optimization. If C is close to zero, then the sole dilatation is taken into account, but if C is of the order of m (the number of I/O units) , only the other criteria are considered. In our mind, C is a coefficient fixed at the system generation for a given site and a given application.

For the moment let all the γ_k be equal to one, for example (all the programs are equivalent).

To tune the coefficient C in order to avoid the appearance of forgotten programs in the system, we performed several experiments. The coefficient C must not be too small so that the forgotten programs are effectively favoured, but not too large so that the dilatation is any more considered.

The best value we have found for C is 1.5 : the system performance is not changed but the users are better served. We give in figure 9 the results we obtained in the same conditions as previously described (see § 4.1).

	Without penalty ($C = 0$)		With penalty ($C = 1.5$)
Number of programs processed during the experiment	52		66
Probability that the system efficiency is < 0.01	Type I	0.47	0.42
	Type II	0.50	0.25
	Type III	0.31	0.20
	Together	0.44	0.31
Mean efficiency of entirely processed programs	0.082		0.123
Mean dilatation during the experiment	3.1		3.1
Number of programs in POOL at the end of the experiment	Type I	19	6
	Type II	20	9
	Type III	11	2
	Total	50	17

Figure 9

4.3 - The other scheduling levels

Other levels or other layers in the system can have their own scheduling policies. Because we consider a level which is very close to the physical resources, the other levels must be able to interact with it. It is the aim of the coefficients γ_k .

Each γ_k ($0 < \gamma_k \leq 1$) is associated with a program k . This coefficient indicates the degradation of services we admit for a program. As a matter of fact, the efficiency of a program k is not directly compared to the system efficiency e_s , but to $\gamma_k e_s$.

We have limited the γ_k 's variations to the interval $]0,1]$ in order to avoid that other scheduling levels can completely ruin physical system performance.

5 - CONCLUSION

This paper deals with the implementation of three principles of optimality for multiprogramming which state the existence of both an optimal multiprogramming degree and an optimal program mixture in the multiprogramming set. Their implementation has been realized in a simulation model which permits, in contrast with analytical models, to relax assumptions on the workload submitted to the system. In the present case, the workload is supposed to be non-stationary and issued from a non-homogeneous program population. Therefore a dynamic control of the system needs special estimators. These statistical problems are studied in detail and an implementation has been realized. The results obtained on the system performance improvement show that the performance can be doubled in the case of a very heterogeneous workload and the "thrashing" avoided in all cases.

Moreover it is noteworthy that the statistical results we derive solve general problems occurring when estimating mean values of non-stationary stochastic processes. Therefore they can be widely used out of the context in which we applied them.

Finally by considering several performance criteria our study is close to the real world. So we are now interested in the implementation of our solutions in a virtual memory system which is being built on a CII-IRIS 80 computer.

REFERENCES

- [1] BADEL M., GELENBE E., LENFANT J., LEROUDIER J., POTIER D. - "Adaptive optimization of the performance of a virtual memory computer" - Proceed. of the Workshop on Computer Architecture and Networks - IRIA - North-Holland Publishing Co. - August 1974.
- [2] BADEL M., GELENBE E., LEROUDIER J., POTIER D. - "Adaptive optimization of a time-sharing system's performance" - IEEE Proceed. on Interactive Computer Systems - June 1975.
- [3] LEROUDIER J., POTIER D. - "Principles of optimality for multiprogramming" - ACM-SIGMETRICS and IFIP W.G. 7.3. Conference Harvard University - March 1976.
- [4] DENNING P.J., KAHN K.C., LEROUDIER J., POTIER D., SURI R. - "Optimal multiprogramming" - Acta Informatica - Vol. 7 - Fasc.2 - Dec. 1976.
- [5] COURTOIS P. - "On the near-complete decomposability of networks of queues and stochastic models of multiprogramming computing systems" MBLÉ Report - Brussels - Belgium - November 1971.
- [6] COURTOIS P.J. - "Decomposability, instabilities and saturation in multiprogramming systems" - Communications of ACM - Vol. 18 - n°7 - July 1975.
- [7] JACKSON J.R. - "Jobshop-like queueing systems" - Management Science, 10, 1, October 1963.
- [8] BUZEN J.P. - "Computational algorithms for closed queueing networks with exponential servers" - Communications of ACM - 16, 9, Sept. 1973.
- [9] CHANG A., LAVENBERG S.S. - "Work-rates in closed queueing networks models" - Operation Research, 22 , 1974.
- [10] POTIER D., LEROUDIER J., BADEL M. - "Un modèle d'analyse des performances d'ordinateurs multiprogrammés à mémoire virtuelle" - Rapport IRIA-LABORIA n° 152 - January 1976.
- [11] LEROUDIER J. - "Systèmes adaptatifs à mémoire virtuelle" - Thèse d'Etat - Grenoble University - May 1977.
- [12] BELADY L.A., KUEHNER C.J. - "Dynamic space-sharing in computer systems" - Communications of ACM - Vol. 12 - n°5 - May 1969.
- [13] BURGEVIN P., LEROUDIER J. - "Characteristics and models of programs behavior" - National Conference ACM 76 - Houston - Texas - Oct. 1976
- [14] BURGEVIN P., INGELS Ph., LEROUDIER J. - "Analysis of program behaviour" - Rapport IRIA-LABORIA n° 237 - June 1977.