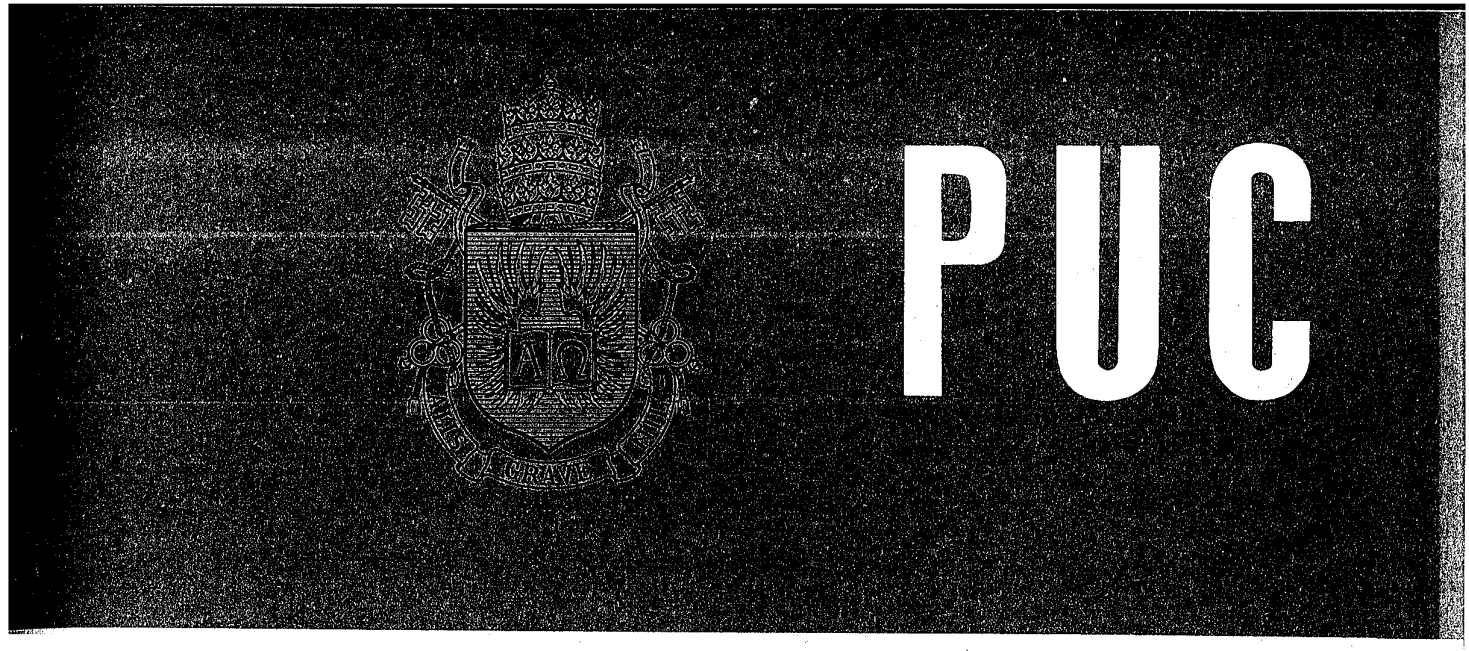


Series: Monographs in Computer Science
and Computer Applications

Nº 5/72

ONE THE USE OF MODELS EMPLOYING BOTH SIMULATION AND ANALYTICAL SOLUTIONS FOR SCHEDULING COMPUTING CENTERS

by

D. P. Bovet

Computer Science Department - Rio Datacenter

Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 209 — ZC-20
Rio de Janeiro — Brasil

ON THE USE OF MODELS EMPLOYING BOTH SIMULATION AND ANALYTICAL SOLUTION FOR SCHEDULING COMPUTING CENTERS

D.P. BOVET

Visiting Professor from the
University of Pisa

This paper was published in the Symposium Computer Simulation Versus Analytical Solutions for Business and Economic Models.

Gothenburg - 28 a 30 August - 1972

Series Editor: Prof. A. L. Furtado September /1972

ABSTRACT

Large scale computing centers are expensive facilities where considerable savings can be achieved through improved scheduling of operations. Several proprietary packages that use both simulation and analytical solutions have been developed to yield optimized schedulings. This paper discusses in some details several approaches that have been taken to solve this problem and explains the trade-off existing between accuracy of the results obtained and costs involved in setting up the model.

1. INTRODUCTION

Computer centers are expensive facilities where conside - rable savings in costs of operation may be achieved through a better utilization of available resources.

This paper is concerned with the problem of scheduling production jobs over a large period of time (tipically one week or one month) for business oriented computer centers.

Production jobs are usually characterized by large execution times, by well known requests for resources (CPU time, core memory, scratch files, tape drives, etc.) and, eventually, by prefixed deadlines for execution (payroll programs to be executed every friday, etc.).

In the early days of computers when jobs were run in a strictly sequential way and when no form of multiprogramming was yet introduced, the scheduling problem was a trivial one because each job execution could be represented as a non overlappable segment on the time axis.

Presently, medium or large scale computer systems include several autonomous processors (tipically CPU and I/O controllers or channels) that are able to execute concurrently several independent jobs or subjobs.

As an example, in a modern multiprogramming system, job A may be using the CPU while, at the same time, job B is using I/O controller 1 to transfer data from disk 3 and job C is using I/O controller 2 to transfer to tape 4.

It should be clear from the previous considerations that, in order to improve the throughput (average number of jobs processed per unit of time) of a multiprogrammed system, it is necessary to execute simultaneously jobs whose requests for resources are as complementary as possible.

In the ideal case, a CPU-bound job should reside in memory with other disk-bound and printer-bound jobs. In practice, however , job composition can vary greatly from time to time and thus the effective progress rate of a job in a multiprogramming environment depends not only on the single job characteristics but also on the characteristics of other jobs currently under execution.

Having briefly reviewed the main problems associated with scheduling jobs in multiprogrammed computer centers, the notion of automatic pre-scheduling is now introduced.

Today the task of pre-scheduling jobs on the input file of the computer is performed in most cases by the operator or by the head of the computer center. This is accomplished manually by examining the resource requirements of each job and its eventual deadline for execution.

Once the pre-scheduling has been performed, the operating system of the computer is responsible for the effective scheduling and attempts on a local scale, that is by scanning a limited portion of the input queue, to select those jobs whose resourse requirements are less subject to mutual interference.

Manual pre-scheduling is adequate or even mandatory in all computer centers where computer load varies impredictably with time and where the list of jobs to be executed on a given day is known only the moment jobs are picked up by the operator to be read in.

In many business oriented computer centers, however, the great majority of jobs are production jobs whose frequency of execution and duration are known a priori. In such cases, manual pre-scheduling has been advantageously integrated if not replaced by automatic pre-scheduling techniques that yield a time table over an extended period of time of the jobs' initiation times.

Ruling out, for obvious reasons of cost, experimentation on computer systems of several possible pre-schedulings, all existing automatic pre-scheduling techniques are based on the use of models of both the jobs to be executed and of the computer system upon which these jobs will be executed.

As a consequence, automatic pre-schedulers include both a scheduling algorithm (SA) and an avaluation algorithm (EA). The scheduling algorithm is based on heuristic techniques to yield at a reduced cost a suboptimal pre-scheduling of jobs: in order to apply it, an evaluation algorithm is used that measures the effective progress rate of the various jobs simultaneously executed on the computer system. During the execution of the SA, the EA signals periodically the names of those jobs whose (simulated) execution is terminated and new job initiation times are inserted by the SA in the time table entries.

Assuming that a suitable evaluation model is available, the SA of a pre-scheduler is based on those operation research techniques developed to solve packing problems (1): the algorithm may be a single

pass or a multipass one and it must include some heuristic rules to select from the many possible sequences of jobs a sequence that satisfies the initial constraints and thus optimizes some objective function like the global execution time or the remaining computer idle time.

A different situation exists with respect to EA where both simulation and analytical models have been successfully used.

The following sections of this paper dicuss alternative approaches used in the design of EA's. Special emphasis is given in section 4 to a mixed approach that uses both simulation and analytical techniques.

2. EA's BASED ON SIMULATION TECHNIQUES

Simulation techniques have been used to develop both general purpose (2) and specialized models (3) of computer systems.

IBM's CSS for example models application programs through alternances of pseudoinstructions requiring CPU time and other pseudoinstructions requiring the use of I/O channels and peripheral devices (see fig. 1)

label 1	process	30
	write	file 1
	read	file 2
	process	50
	wait	
	process	75
	send	file 3
	return	

(fig. 1) "A Detailed Model of Application Program"

The simulation of each pseudoinstruction requires in general several distinct events to model the operating systems interactions and the occurrence of I/O interrupts.

CSS like other computer simulators aims at an accuracy of the model rather than at a reduced cost both in terms of development and use. For that reason, these models are more appropriate as tools to predict the effects of software or hardware modifications in a computer system than as Evaluation Algorithm for automatic pre-schedulers.

3. EA's BASED ON ANALYTICAL TECHNIQUES

Simple analytical models have been developed to measure the effective overlapping capabilities between I/O and CPU of a computer system with respect to a single program.

Auerbach's reports (4) for instance use a rough technique to determine the total execution time of benchmark programs based on the CPU instruction timings and on the execution times of the I/O routines. Total execution time is then determined as the maximal between CPU and I/O operations execution times.

A more detailed treatment of this problem can be found in (5).

No satisfactory analytical technique has been developed for measuring the execution time of programs in a multiprogramming environment.

An exception is constituted by time sharing systems where users require limited amount of CPU time and where most I/O activity is limited to page swapping between main and secondary storage. In such cases, keeping also into account the considerable delays involved in data transmission and the frequent updating commands used by time sharing users, it is possible to model users' interactions with the system with few random variables representing the interarrival rate of commands and the number of quantum of time required and to derive the global response time of the system for a given traffic. A comprehensive review of analytical time sharing models is found in (6).

4. EA's BASED ON SIMULATION AND ANALYTICAL TECHNIQUES

In recent years, considerable attention has been paid to mixed EA's using both simulation and analytical techniques that allow the determination of throughput of a multiprogrammed computer system for a given program load.

Unfortunately, these EA's are imbedded in proprietary programs of software companies and the available documentation is rather succinct.

In this section an approach similar to the one used by SCERT (3) is outlined.

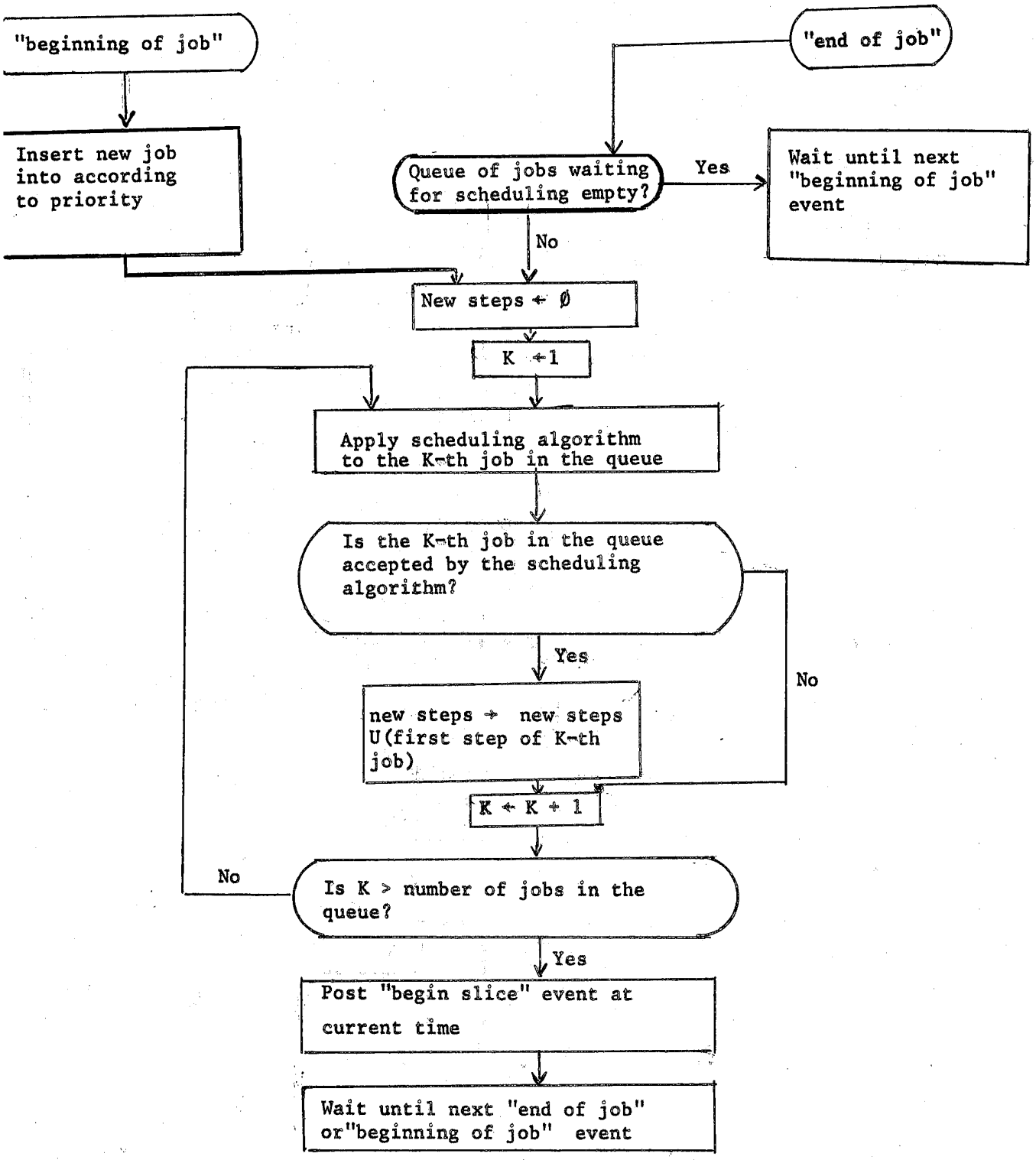
A first simplification made by SCERT and by other similar techniques consists of modelling application programs through two distinct sets of values: CPU requirements and I/O requirements. CPU requirement expressed in seconds may be obtained, knowing the instruction execution times for a given CPU, by estimating the number and

the type of CPU instructions corresponding to a program execution. Similarly, I/O requirements are obtained by describing the number of records accessed for each file. Secondary storage access time and supervisor overhead are then added separately to the pure data transmission time to derive the total time spent in accessing each distinct file. Finally, using a technique similar to the ones illustrated in section 3, the total execution time of a program in a uniprogramming environment is derived. Simulation of runs in multiprogramming proceeds according to the following rules: during the simulation of an operating interval (e.g. an 8 - hour shift), a critical path list is built of all application programs that are condidates for execution during that interval.

Each application program is described as a partially ordered set of jobs and each job is executed as a sequence of steps.

The critical path list which is constantly updated during the simulation contains the names of all the jobs which remain to be executed in order to terminate the corresponding application programs. Each job in the list is characterized by an earliest starting time , that is the earliest time at which all jobs preceding it have been completed and a latest starting time , that is, the latest time before which it is possible to execute the job and all its successors and still terminate the execution of the application program before the deadline established by the user.

Besides deadline constraints, jobs are ranked by priority and time of arrival. Each time a new job enters the system, it is inserted in the queue of jobs waiting for scheduling and a new job scheduling is attempted according to the procedure illustrated in fig. 2.



*ACTIVITY SCANNING THE QUEUE OF JOBS WAITING FOR SCHEDULING"

FIGURE 2

The scheduling algorithm itself that decides whether a given job will pass the job scheduling test performs the following verifications on the job:

- are they too many jobs already scheduled (some systens have an active job list of finite size)?

- have all predecessors of that job already been executed?

- is it possible to run that job and all its successors within the given operating interval?

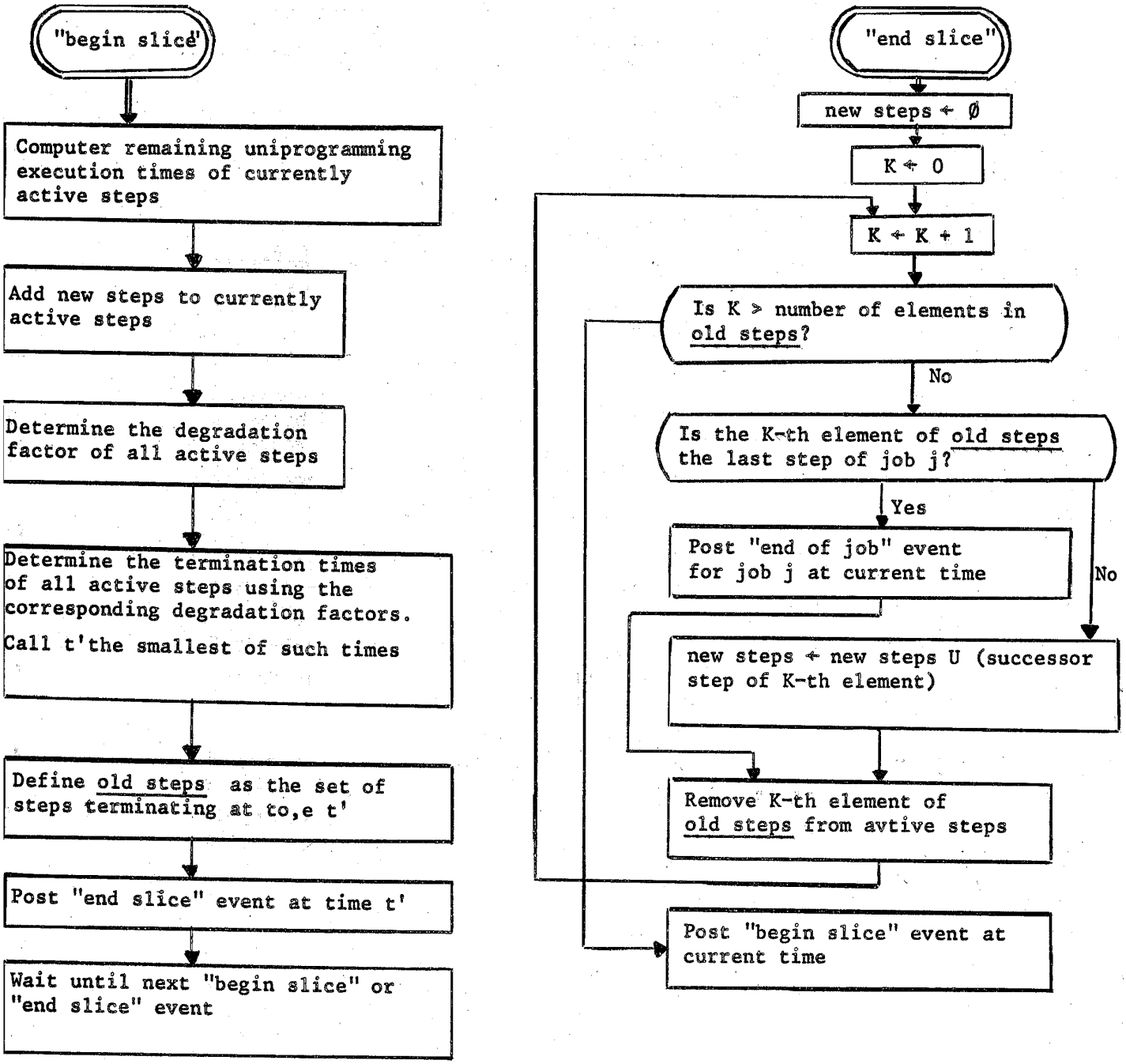
- has the latest starting time of the job been reached?

- will be introduction of that job produce interference with others previously scheduled jobs so that one or more of them could become critical (there is a possibility that they won't terminate before the end of the operating interval)?

If a job passes successfully the job scheduling test, it becomes active and starts competing for the resources of the system.

The execution of steps corresponding to a given job is illustrated in fig. 3. For reasons of simplicity, discrete simulation techniques are not used and an analytical formula is used to determine the degradation factor of each active run during a given time slice.

The basic idea underlying the degradation factor concept is the following: if run j_{11} requires 2200 msec of execution time in uniprogramming, run j_{21} requires 1500 msec, and run j_{31} requires 1300 msec, then the simultaneous execution of j_{11}, j_{21}, and j_{31} in a multi-



"ACTIVITY DETERMINING THE TIME SLICES DURATIONS"

FIGURE 3

programming environment can be estimated during the first time slice of 2730 msec by stating that, due to the hardware and software characteristics of the system considered and to the competition for the same resources, j_{11} will have a degradation factor of 2.5, j_{21} a degradation factor of 2.9, and j_{31} one of 2.1. In other words, during the first 2730/2.5 of its total run time of 2300 msec and the remaining uniprogramming execution time after the first slice is equal to:
2300 - 2730 / 2.5 = 1108 msec.

The algorithm to compute the degradation factor is probebly the most delicate and most confidential part of EA's, although it is not hard to imagine some of the principles underlying its structure.

5. CONCLUSIONS

The internal structure of automatic pre-schedulers has been briefly discussed in the previous sections. It appears at the present time that a mixed approach that uses both simulation and analytical solutions is the only feasible one when dealing with complex multiprogrammed computer systems.

Experience deriving from using automatic pre-schedulers has already yielded some interesting conclusions: predictions ob - tained by using the model differ from 10% to 20% with respect to actual execution times for typical batch operations and the execu- tion time of the model itself has been kept at a reasonable level.

New problems are actually coming to the surface with the introduction of commercial real time multiprogramming systems: in such cases the approach outlined has been proved to be inaccurate and relatively expensive due to the larger number of events to be simulated and other techniques will have to be found to deal with these new problems.

REFERENCES

(1) A. R. Brown - "Oprimun Packing and Depletion"
Mc Donald/American Elsevier Computer Monographs 1971

(2) O. J. Dahl - "Discrete Event Simulation Languages"
Programming Languages p. 349-395, F. Genuys editor, Academic Press 1968.

(3) J. Le Foll - "Problèmes d'évaluation des systèmes: la méthode PRESTE" , METRA, nº 1 p. 27-55, 1971.

(4) Auerbach INFO, Inc. - Standard EDP Reports, Vol. 1.

(5) H. Hellerman, H. J. Smith, Jr - "Throughput Analysis of Some Idealized Input, Output, and Compute Overlap Configurations"
Computing Surveys, vol. 2 nº 2, p. 111-118, 1970.

(6) J. M. Mc Kinney - "A Survey of Analytical Time-Sharing Models"
Computing Surveys, vol. 1, nº 2 p. 105-116, 1969