An analysis, both theoretical and by simulation, of a time-shared computer system

By J. P. Penny*

The complexity of operation in a computer system makes difficult the task of examining its behaviour theoretically. In a study made here of a time-shared computer system, the author shows, however, that a fairly simple theoretical analysis can be particularly valuable as an adjunct to a simulation study.

1. Purpose

Before a time-shared computer system is constructed, the designer should have the answers to a number of questions. He should know, for example:

(i) the extent of the improvement likely to be gained by time-sharing,
(ii) the factors which will limit the degree of improvement,
(iii) the effects on the degree of improvement of any modifications which are seen to be feasible.

In practice, the answers to these questions, if found at all, are usually found by field testing when the time-shared system is actually in operation.

The improvement in processing capacity gained by time-sharing between a number of programs depends on many complex factors: for example, the average demand and pattern of demand for time by each program, the combinations in which programs happen to be run, and the procedure used for time-sharing. Though simple enough as conceived initially, the time-sharing procedure itself may become quite involved. Its detailed development and practical effects are subject to many limitations of the computer configuration and the soft-ware used.

The number and complexity of the factors involved make it impossible to obtain highly accurate results by theoretical analysis. Accurate results are particularly necessary if the effects of minor modifications to the time-sharing procedure are to be evaluated. One is therefore virtually compelled to undertake a computer simulation study to obtain all the information that is needed.

If a simulation study must be made, it might be argued that the need for any theoretical calculations disappears. The chief aim in the paper is to show that a theoretical study, particularly if made *in conjunction with a simulation study*, can be extremely worthwhile. A theoretical analysis can:

(i) give valuable preliminary information which may be used to determine the scope of the proposed system,
(ii) reduce the quantity of work in the simulation study by making it possible to extrapolate fairly extensively from the results of a very few simulations,
(iii) suggest strategies, to be checked by simulation, which may be followed to increase the efficiency of the system.

These points will be illustrated by an analysis of a specific type of time-shared system. Upper and lower limits will be found for the "improvement factor", a measure of the increase in processing capacity to be gained through time-sharing. These limits are expressed as functions of parameters which may be estimated for a particular computer and work-load.

2. The model

The author developed and brought into operation a multi-program system for the low-cost computer, CIRRUS (Penny and Pearcey, 1962; Penny, 1963). The analysis made here is directed towards CIRRUS, but should be to some extent applicable for other systems which, like CIRRUS, use only one level of storage. Designers of time-shared systems have perhaps more often used a second level of storage, usually a disc file or drums. However, really large core stores of as many as half-a-million words are now becoming available at not excessive costs. The simplicity and inherent efficiency of the system which uses only one level of storage could well attract more designers to this configuration.

In the type of system under discussion, operations within any one program are usually performed in an order which is unchanged from execution without time-sharing. This characteristic, which greatly simplifies the task of analysis, distinguishes the one-level from the two-level store system. In Atlas (e.g. Kilburn *et al.*, 1961), an example of the latter class of system, information from slow peripherals is fed through core to the backing store in advance of its being required by the program or programs using processor time. Information to be output is also transferred to the backing store when it is produced at a rate faster than that at which it can be handled.

The aim in building the CIRRUS system was that the computer be made "to behave as a set of separate and independent computers" (Penny and Pearcey, 1962).

* *C.S.I.R.O. Computing Research Section, Canberra, Australia.*

The work-load is supplied through "operating stations", each equipped with a fast paper-tape reader and punch, an operator's keyboard and a monitor typewriter. These stations are at present identical, but this feature is not a pre-requisite for the validity of the analysis. Programs held in the 32,000-word store "compete" for processor time. The present system allows for three stations. The results of the analysis made here could be used to determine, amongst other things, the economics of providing more storage and peripheral units to allow for a greater number of programs.

CIRRUS is perhaps unusual in at least one respect. The 3,000-odd instructions constituting the compiler are, together with standard routines such as those for input and output, stored permanently near the end of the store. A technique of "parametric addressing" (Penny and Pearcey, 1962) allows these instructions to be used independently by separate programs. This technique appears to give routines the property of "re-entrance" as proposed by Radin and Rogoway (1965). As a result, compilation of one program can take place simultaneously with compilation or execution of others.

The operation of a program from any given station therefore parallels its operation in the system not time-shared, except that processor time is needed before compilation begins and after execution terminates for "space (i.e. store)-sharing" decisions. During operation, of course, the program may be delayed when processor time cannot be made available, and some time is also lost in making time-sharing decisions. The effect of both space- and time-sharing "overhead" is allowed for in the analysis.

The number of programs which can be held in store is obviously variable. To allow for the effects of storage limitations in the analysis, one would have to know the distribution of storage requirements by programs in the work-load and the correlation of program size with running time. There is probably also a significant correlation between program size and the proportion of processor time used, since programs of many instructions may have relatively less input and output than smaller programs. None of these factors is easy to estimate.

In making the analysis, it is assumed that the number of programs held in store remains constant during processing of the work-load. It cannot therefore be said that the quantity of work done by the time-shared system would exceed by some definite factor that done without time-sharing. If one wishes to find some definite value which takes account of store limitations, the answer is better sought through simulation. Even then the difficulties are substantial. A sample of programs must be examined to build a "work-load" for the simulator. To account fully for all the relevant factors, the number of programs examined must be very large and the examination made in considerable detail.

However, one of the chief aims of any preliminary study must be to obtain information which will help in determining the size of the projected system. Such decisions must be based on a comparison of the improvement to be gained in a system of some given size against the cost of constructing the system. If one knows the improvement gained through sharing between n programs, say, and the improvement for $n + 1$ programs, a decision can be made on whether a proportionate increase in storage and peripheral equipment is justified.

3. The analysis

Consider first the situation where the processor is not time-shared. We can take T, the total time required to process a given work-load, as a quantitative measure of the "work" in the work-load. T can be divided into alternate segments T_i, t_i, where

T_i is a period during which the processor is used continuously,

t_i is a period, following T_i, in which the processor is not used.

The degree of processor utilization is therefore:

$$D = \frac{\sum_i T_i}{\sum_i (T_i + t_i)} = \frac{\sum_i T_i}{T}.$$

Delays by the computer operator contribute to Σt_i, and therefore to T. Operator delays in single-program operation must certainly be taken into account in any comparison against multi-program operation.

Suppose that, by time-sharing, the time required to process the work-load becomes T'. The capacity of the system to process the work-load has therefore been improved by a factor I, where

$$I = \frac{T}{T'}$$

For example, if the time required is halved, processing capacity has obviously been doubled.

Suppose also that work is supplied to the processor in n streams $S_1, \ldots S_n$, and that each stream, if allowed exclusive access to the processor, would use a proportion d_j, $j = 1, \ldots n$, of total processor time. An obvious upper limit of I is therefore

$$I_U = n.$$

Now,

$$I = \frac{\sum_i (T_i + t_i)}{T'} = \frac{\sum_i (T_i + t_i)}{\sum_i T_i} \cdot \frac{\sum_i T_i}{T'} = \frac{D_T}{D}$$

where D_T = processor utilization when the processor is time-shared.

Hence,
$$I'_U = \frac{1}{D} \qquad (3.1)$$

is another upper limit for I. In practice, processor time available for the work-load is reduced by "overhead", or time spent in implementing time-shared operation. The computer structure, time-sharing method, composition of the work-load and the distribution of the work-load between streams will all affect overhead time.

For a particular multiprogram computer and work-load, let us express overhead as $\phi(n)$, where $\phi(n)$ is a proportion of total processor time. (3.1) can therefore be replaced by

$$I'_U = \frac{1 - \phi(n)}{D}. \tag{3.2}$$

In establishing a lower limit for the improvement factor, let us assume that there are two constraints:

(1) time is shared according to a fixed order of priorities between streams, work in stream S_j taking precedence over work in stream S_{j+1},
(2) there can be concurrency of useful work not requiring processor time with work requiring time only where the former takes place in the higher priority stream.

Manipulation of priorities can be a valuable tool to increase the efficiency of the system. One of the objectives in simulation will be to test the effect of different methods of priority variation. However, the method of priority adjustment may become quite complicated. In the theoretical analysis, it is preferable therefore to assume that a straightforward and constant order of priority exists.

Work not requiring processor time will often be done while the processor is busy with work in a higher priority stream. It must be remembered that periods spent waiting for a peripheral unit or spent in operator delays contribute to the quantity of useful "work". Work not requiring processor time will continue in a low priority stream whenever the period of work has begun before control must be returned to a higher priority stream. The frequency and extent of this overlapping is difficult to predict theoretically. It is therefore preferable to ignore the possibility and to assume that the effect of this concurrency can be measured by simulation.

The imposition of each of the two constraints would reduce the quantity of work done. Any value found for the improvement factor in a system where the constraints hold can therefore be taken as a *lower limit* for the improvement factor in a more general case.

Let us suppose that the two constraints hold, and consider first an hypothetical case where there is no overhead time. In stream S_1 alone, a quantity of work will be processed equal to that done without time-sharing. Furthermore, a proportion $(1 - d_1)$ of total time will be shared between work in lower-priority streams. Where $j > 1$, each stream will have highest priority use of a proportion

$$\prod_{k=1}^{j-1}(1 - d_k)$$

of total time, and will leave a proportion

$$\prod_{k=1}^{j}(1 - d_k),$$

again of total time, free for lower priority streams.

The improvement factor for this hypothetical system in which $\phi(n)$ is zero is

$$I_{H,\,\phi(n)=0} = 1 + \sum_{j=2}^{n}\prod_{k=1}^{j-1}(1 - d_k).$$

Let the time required to process the work-load in this system be T''.

i.e. $$I_{H,\,\phi(n)=0} = \frac{T}{T''}.$$

In practice, some overhead time must certainly be spent. Of the time T' required to process the work-load, let θ be the actual time spent in overhead.

Now, $$T' \leqslant T'' + \theta.$$

Inequality, rather than equality, will hold if any time spent in overhead coincides with a period in which the work in no stream would have used the processor.

Therefore, $$I_{H,\,\phi(n)\neq 0} \geqslant \frac{T' - \theta}{T'} \cdot \frac{T}{T''}$$

i.e. $$I_{H,\,\phi(n)\neq 0} \geqslant \left(1 - \frac{\theta}{T'}\right) . I_{H,\,\phi(n)=0}.$$

Now, $$T' \geqslant \frac{T}{n}, \text{ so}$$

$$I_{H,\,\phi(n)\neq 0} \geqslant \left(1 - \frac{n\theta}{T}\right) . I_{H,\,\phi(n)=0}. \tag{3.3}$$

The right-hand side of (3.3) is a lower limit for the improvement factor in the restricted case where the two constraints stated hold. It is therefore a lower limit for the general case. Thus, the improvement factor lies in the range:

$$\left(1 - \frac{n\theta}{T}\right)\left\{1 + \sum_{j=2}^{n}\prod_{k=1}^{j-1}(1 - d_k)\right\} \leqslant I \leqslant \text{Min}\left\{n, \frac{1 - \phi(n)}{D}\right\}.$$

Now, $$\frac{1 - \phi(n)}{D} = \frac{1 - \theta/T'}{D} \leqslant \frac{1 - \theta/T}{D}.$$

Therefore,

$$\left(1 - \frac{n\theta}{T}\right)\left\{1 + \sum_{j=2}^{n}\prod_{k=1}^{j-1}(1 - d_k)\right\} \leqslant I \leqslant \text{Min}\left\{n, \frac{1 - \theta/T}{D}\right\}.$$

Overhead θ can be considered in two separate parts:

(1) space-sharing overhead, processor time spent in assigning storage and peripheral units,
(2) time-sharing overhead, processor time spent in switching from one activity to another.

It is probable that, for any multiprogram computer, the quantity of overhead time in processing a given work-load can be expressed in terms of the number of programs and quantity of input and output in the work-load. Suppose that, for each program to be processed, a period s such that

$$s_1 \leqslant s \leqslant s_2$$

is needed for space-sharing decisions. Suppose also

that, for each unit* of information to be transferred, a period t such that

$$t_1 \leqslant t \leqslant t_2$$

is spent in time-sharing decisions.

If, in the work-load there are p programs and m units of information to be transferred, then θ lies in the range

$$ps_1 + mt_1 \leqslant \theta \leqslant ps_2 + mt_2.$$

Hence,

$$\left(1 - \frac{n(ps_2 + mt_2)}{T}\right) \cdot \left(1 + \sum_{j=2}^{n} \prod_{k=1}^{j-1} (1 - d_k)\right) \leqslant I \leqslant \text{Min}\left\{n, \frac{1 - \frac{ps_1 + mt_1}{T}}{D}\right\}. \quad (3.4)$$

For a particular computer and time-sharing procedure, t_1, t_2, s_1 and s_2 should be calculable with reasonable accuracy. For a given work-load run on this computer, T, m and p are constants and estimates for them should also be calculable. Finding an estimate for D, the overall degree of processor utilization by the work-load, should also be feasible. All that is needed, then, is to relate the values of d_k to D. I_U and I_L would then be expressed in terms of only one variable n, the number of separate streams.

4. Estimating overhead

The most important result to be found is the improvement in processing rate to be gained by time-sharing between a given number of programs. Processor utilization by the work-load without time-sharing (D) is the most important factor. The calculation is very much simplified if an estimate can be made for a range of θ/T. In a system using only one level of storage, the quantity of time used in overhead should be reasonably small, but will certainly be significant.

Reasonable estimates of ranges for s and t can probably be made once the proposed system has been outlined. Let us take CIRRUS as a specific case. The computer is of course now in operation. The quantities of time needed for space- and time-sharing decisions are known, and agree fairly closely with early predictions. It has been found that less than one second per program is required for space-sharing decisions,

i.e. $$0 \leqslant s \leqslant 1.$$

The time-sharing system requires a switch between programs for each *character* of information transferred with any peripheral, unless the transfer can be made immediately. For calculating time-sharing overhead, a single character is conveniently taken as the "unit" of information transferred. The period of time lost as time-sharing overhead for each character is as little as 12 μsec if the character can be transferred immediately—only an availability check is made. If the character cannot be transferred, as much as 184 μsec* can be lost. This is the maximum time for two program switches; first to a lower priority program and then followed by a later return when the transfer can be made.

Therefore $$12\ \mu\text{sec} \leqslant t \leqslant 184\ \mu\text{sec}.$$

To estimate p, m and T, something must be known about the likely work-load. For use in his simulation study, the author selected at random a sample of ten programs from those programs being run on the two computers (an IBM1620 and an IBM7090) then available to programmers in the University of Adelaide. These programs were examined in logically separable sections (and frequently with data array sizes reduced) under a tracing program on the IBM1620. This program gave a count of the (1620) instructions executed between successive branches to the input–output subroutine, together with a statement of the quantity of information involved in the input or output operation.

By considering in detail a number of small subroutines, an estimate was made for a conversion factor by which each instruction count should be multiplied to find the appropriate period of continuous processor usage in CIRRUS. The behaviour of CIRRUS during sequences of input and output with each peripheral unit was also predicted. Hence, it was possible to express for the simulator each of the ten programs as a sequence of alternate periods T_i, t_i, of processor use and waiting time, as suggested at the beginning of Section 3.

If each program were to be compiled once and assembled once from object code, this "work-load" would have the following parameters:

$$p = 20$$
$$m = 3 \cdot 10^5 \text{ characters}$$
$$T = 3 \cdot 10^3 \text{ seconds}$$

Therefore, $$0{\cdot}001 \leqslant \theta/T \leqslant 0{\cdot}027.$$

Suppose that, generally,

$$0 \leqslant \theta/T \leqslant 0{\cdot}03.$$

Substituting in (3.4), we obtain

$$(1 - 0{\cdot}03n)\left(1 + \sum_{j=2}^{n} \prod_{k=1}^{j-1} (1 - d_k)\right) \leqslant I \leqslant \text{Min}\left\{n, \frac{1}{D}\right\} \quad (4.1)$$

5. Information obtainable from the analysis

Before useful results can be obtained, it is necessary to relate the d_k to D. Again taking CIRRUS as a particular case, assume that each "stream" as specified in Section 3 is constituted of the work supplied through a single station. Suppose also that the work-load is very large and is divided randomly between stations. Since the stations are identical, $d_i \to D$ for all i. If the

* Any convenient quantity.

* The maximum period of time lost depends on the number of programs operating. The period quoted is the maximum when there are four programs, four being the greatest number considered here.

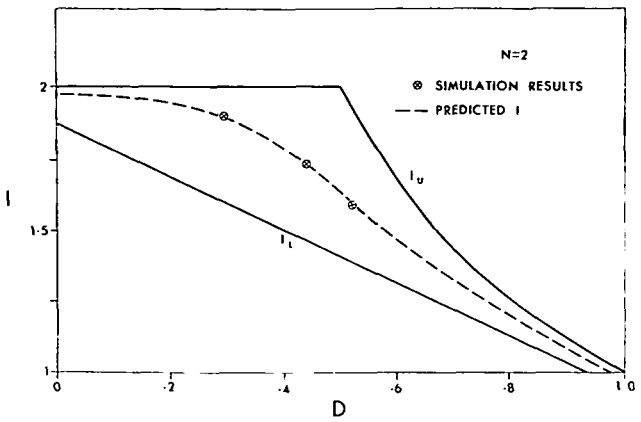

Fig. 1.—Upper and lower limits I_U, I_L, for the "improvement factor" I, plotted against processor utilization D: Two-program case

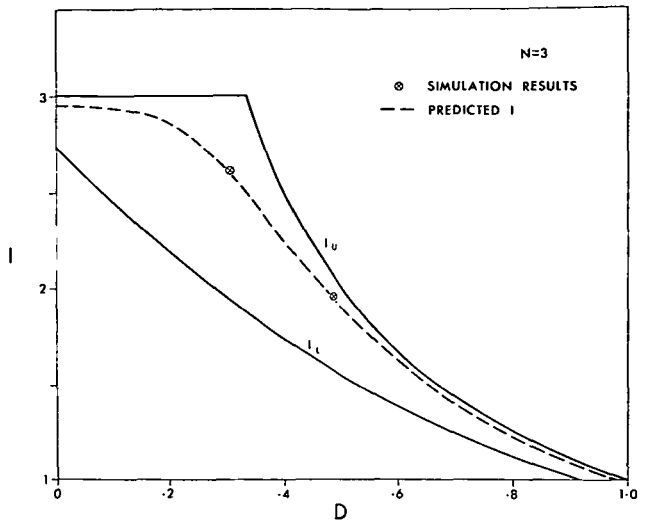

Fig. 2.—Upper and lower limits I_U, I_L, for the "improvement factor" I, plotted against processor utilization D: Three-program case

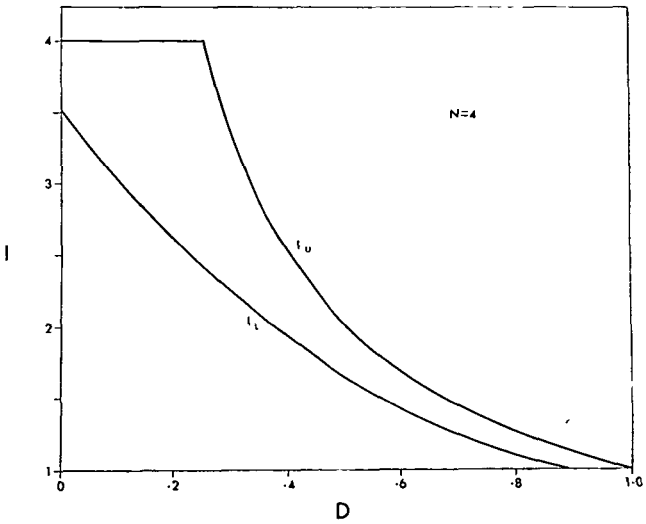

Fig. 3.—Upper and lower limits I_U, I_L, for the "improvement factor" I, plotted against processor utilization D: Four-program case

work-load were not divided at random, it can be assumed that the "scheduling" carried out would be aimed at improving the rate at which the work-load was processed. D can therefore, in this case, be substituted for all d_i in the expression for I_L in (3.4).

Then,

$$(1-0{\cdot}03n)\left(1+\sum_{j=1}^{n-1}(1-D)^j\right)\leqslant I\leqslant \mathrm{Min}\left\{n,\frac{1}{D}\right\}. \quad (5.1)$$

From (5.1) upper and lower limits of I can be calculated for particular values of D and n. I_L and I_U for $0 < D < 1$ are plotted in **Figs. 1, 2** and **3** for $n = 2$, 3 and 4, respectively.

It is clear that a reasonable estimate must be made of the degree to which the processor would be utilized without time-sharing. If the sample set of ten programs prepared for the simulation study were each run twice, being compiled once and assembled once, D for this "work-load" would be

$$D=\frac{\sum_{\text{all programs}}\ \sum_{\text{each program}} T_i}{\sum_{\text{all programs}}\ \sum_{\text{each program}} (T_i+t_i)}$$

$$=\frac{818\ (\text{sec})}{2792\ (\text{sec})}=0{\cdot}29. \quad (5.2)$$

This estimate would neglect the effect of delays by the computer operator between or during program. If, for example, the mean idle time per program were t seconds, $20\,t$ should be added to the denominator in the left-hand side of (5.2). If t were 10 seconds, processor utilization would be

$$D = 0{\cdot}27.$$

The number of programs in this sample is admittedly small. However, it is reasonable to say that processor utilization during operation without time-sharing would be fairly low, and quite possibly 0·3 or less. The improvement gained by time-sharing is therefore certainly significant.

For example, if $n = 2$, $D = 0{\cdot}27$,
$1{\cdot}64 \leqslant I \leqslant 2$, (from 5.2)
if $n = 3$, $D = 0{\cdot}27$,
$2{\cdot}06 \leqslant I \leqslant 3$,
and if $n = 4$, $D = 0{\cdot}27$,
$2{\cdot}33 \leqslant I \leqslant 3{\cdot}7$.

It is interesting to compare the order of the improvement to be gained through time-sharing against the improvement obtainable by increasing processor speed.

If the speed of the processor used as a basis for comparison were increased by some factor k, processing capacity would be improved by a factor

$$I' = \frac{1}{(1 - D) + D/k}.$$

For example, if $D = 0{\cdot}27$ as estimated from the sample set of programs, increasing only processor speed could improve processing capacity of the system by, at most, $1{\cdot}37$ times.

The expression for the lower limit of the improvement factor in (3.4) suggests that higher priority should be given to activities requiring a lower proportion of processor time. That this should be done is generally accepted to be worthwhile. In the first simulation run, however, no attempt was made to assess priorities on this basis. A case of 3 programs was simulated, using the sample set of programs (each being run at least twice) as a work-load. For $n = 3$, $D = 0{\cdot}29$ (no operator delays assumed), $I = 2{\cdot}40$ was found, compared to the expected range of $2{\cdot}09 \leqslant I \leqslant 3$. Since the first constraint of Section 3 still applied, it was assumed that the absence of the second constraint caused the increase in the value of I beyond the lower limit found theoretically.

To increase efficiency in practice, two steps can be taken. First, jobs may be scheduled to be run in suitable combinations and second, priorities may be adjusted along the lines suggested in the preceding paragraph.

Elaborate scheduling procedures are important parts of some systems (for example, the Honeywell-800 (Honeywell, 1961) and IBM Stretch (Codd, 1960) systems). The aim in these cases is to associate jobs whose requirements, both in space and time, are complementary. For a formal scheduling procedure to be effective, however, the storage requirement, execution time and processor utilization of each program must be known fairly accurately. Hence, scheduling is of most value when the work-load contains a high proportion of recurring jobs.

Where a machine is used for "open-shop" scientific computing, the work-load varies greatly from day to day. Many jobs are checked-out and then run only once or twice before being discarded. Scheduling for CIRRUS in particular would have been further complicated by the fact that each operator should preferably be able to ignore completely the existence of others. Hence, the potential benefits of any scheduling system had to be carefully balanced against the problems of implementing it.

It is obvious from Figs. 1, 2 and 3 that, unless the value of D is high, the rate of processing will be very much greater when $n = 2$ than when $n = 1$, and much greater again when $n = 3$ rather than 2, or 4 rather than 3. Certainly, any scheduling procedure which materially increases the average number of programs held in store would be worthwhile. The value of I ($2{\cdot}40$) found in the first simulation run where $n = 3$ was appreciably below the upper limit ($3{\cdot}0$). However, no attempt at priority adjustment had been made.

Any method for priority adjustment should be "automatic", that is, no information should be demanded from either the programmer or operator. A scheme was developed whereby the multiprogram control software could reassess priorities whenever the nature of the activity on any station altered. It was assumed from (3.4) that the aim should be to minimize processor utilization by the work constituting the highest priority stream. It was found to be important not only to give higher priority during sequences of input or output but also to make choices between programs engaged in sequences of computation.

Three further simulation runs were made for the same initial conditions. The results of each run confirmed the view that any measure which reduced d_1 relative to d_2, and d_2 relative to d_3 would increase the value found for I. The best result obtained ($n = 3$, $D = 0{\cdot}29$) was

$$I = 2{\cdot}61.$$

So far, only the particular case where $n = 3$, $D = 0{\cdot}29$ has been discussed. Each simulation run required a considerable amount of computer time—almost an hour each on the IBM7090. The value of the theoretical analysis in reducing the quantity of simulation now becomes apparent. By altering certain parameters, the work-load for the simulator was given an apparent processor utilization value D in the vicinity of $0{\cdot}5$. With only two simulation results for the three-program case, a complete curve could be drawn to predict values of I for all D (Fig. 2). Three runs were made for the two-program case and the curve for I against D drawn (Fig. 1). Although no runs were made for the four-program case, the shape of the curve for I against D can readily be foreseen.

These simulation results confirm the view that the average number of programs in store must be kept as high as possible. However, scheduling or selection of programs on any other basis, such as the association of jobs of low and high demand for time, can quite possibly be neglected. Except under certain conditions (for example, where $n = 2$, D for the work-load $= 0{\cdot}5$), no great improvement would be gained. It must be remembered that the curves drawn for I_U in Figs. 1, 2 and 3 allow for no overhead time at all. The author therefore suggested the scheduling procedure which will now be briefly described. The procedure is aimed solely at making effective use of store space in a system having independent consoles and used in an "open-shop" environment.

For the type of scheduling procedure put forward by Codd (1960), both the storage requirement and execution time of each program must be known quite accurately. The storage requirement can be estimated fairly accurately in advance or found exactly when the program is compiled. Furthermore, the storage requirement will remain constant unless the program is altered. On the other hand, execution time is difficult to predict and may vary over successive runs with different sets of data. If the

forecast of execution time were underestimated by a factor of 2 (as would frequently happen in an "open-shop" environment), a schedule prepared by the method suggested by Codd could become meaningless.

If the scheduling procedure is to be effective for the work-load expected in an "open-shop" installation, the procedure must be "dynamic". In other words, selection of the appropriate job should be made only when a new job is needed, and made according to the current conditions. Two obvious requirements, therefore, are that the estimated (or known) storage requirement of each job be stated, and that the quantity of vacant store space be specified both when a job ends or when requested by an operator. Each operator must also be able to reject any job which, when loading is attempted, proves too large. All these facilities are simple to provide.

The author anticipated that at least one of the CIRRUS operating stations would be manned by a full-time operator who would attempt to maintain a steady flow of production jobs to the machine. The on-line user will of course choose any job which happens to be convenient and which fits into the store. The full-time operator should, however, accept as his task the most efficient use of store. If, for example, there are programs in store for all stations but his own, he should choose from those programs available the largest which can be fitted into store. On the other hand, if there are no programs in store, he might choose a quite small program, thereby leaving as much space vacant as possible for other users.

6. Conclusions

The example given illustrates in particular the advantages of carrying out both a theoretical and a simulation study concurrently. The sample set of programs prepared for the simulation study yielded values of p, m and T which enabled an estimate to be made for the "overhead" term θ/T. The value for processor utilization D found for this sample gave some indication of the area in which I_U and I_L should be evaluated. Hence, even before attempting simulation, it was possible to decide whether a system of a particular size could be justified.

The value of the theoretical analysis in enabling complete curves to be drawn after very few simulation runs is self-evident. The search for an appropriate method of priority adjustment was based on attempts to increase I_L in expression (3.4) which was derived by theoretical analysis.

The information yielded on the question of scheduling was particularly valuable because both the structure of the computer and the nature of its expected work-load raised obstacles against scheduling. The scheduling, or more properly "selection", procedure suggested, although primitive by comparison with procedures in some other systems, represents a reasonable compromise for this difficult environment.

The author is well aware that the time-shared computer will not be operated in the same manner as the computer not time-shared. On-line users in particular can be expected to "waste" a good deal of time. However, the important point is that, if the improvement factor for a three-station system is 2·61 (as it is when $D = 0{\cdot}29$), one has while three programs can be held in store the processing capacity and *operating convenience* of three separate computers, each working at a rate 0·87 times as fast as the single computer not time-shared. One can certainly then feel that each of these "separate computers" might be used in a much more relaxed fashion—this is after all one of the chief aims in time-sharing.

The results obtained and the conclusions drawn here apply directly only for a system in which the work-load is supplied through separate stations. The results may have some validity, however, for other systems which, although not having individual stations, make use of only one level of storage. The "streams" into which the work-load is considered to be divided need not emanate from quite separate sources. The crucial factor for the success of the analysis is that the execution of operations within each program must occur in the same order that they would in a computer which is not time-shared.

References

CODD, E. F. (1960). "Multiprogram Scheduling," *Comm. Assoc. Comp. Mach.*, Vol. 3, p. 347 (Part I), p. 413 (Part II).

KILBURN, T., HOWARTH, D. J. H., PAYNE, R. B., and SUMMER, F. H. (1961). "The Manchester University Atlas Operating System," *The Computer Journal*, Vol. 4, p. 222.

Minneapolis-Honeywell Regulator Company (1961). *Executive System Manual for the Honeywell 800.*

PENNY, J. P. (1963). "The CIRRUS Multiprogram System," *Proc. Aust. Comp. Conf.*, Melbourne, March 1963.

PENNY, J. P., and PEARCEY, T. (1962). "Use of Multiprogramming in the Design of a Low-cost Digital Computer," *Comm. Assoc. Comp. Mach.*, Vol. 5, p. 473.

RADIN, G., and ROGOWAY, H. P. (1965). "NPL: Highlights of a New Programming Language," *Comm. Assoc. Comp. Mach.*, Vol. 8, p. 9.