COSC460
RESEARCH PROJECT
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF CANTERBURY

A SYNTHETIC MEASURE OF RESPONSE TIME

C.R.SHEEDY
NOVEMBER 1978

CONTENTS

1. Introduction 4
2. Background 7
2.1 Response Time 7
2.2 Benchmarking 9
2.3 System Monitors 10
3. A Suggested Response Time Measure 12
3.1 The Proposed Method 12
3.2 Resources Used by a Sample Program 15
3.2.1 CPU 15
3.2.2 Discs 16
3.2.3 Main Memory 16
3.2.4 Other Resources 17
3.3 Priorities 18
4. An Implementation of the Proposed Measure 19
4.1 The Computer System 19
4.2 The Synthetic Program 19
4.3 The Resources Used 21
4.3.1 CPU 21
4.3.2 Discs 21
4.4 Results 24
5. Summary 26

Acknowledgements 26

References 27

Appendices

1. General Equations for the Synthetic Program 29
2. Response Time Measurement Program for the PDP 11/70 31
3. Examples of Equations for the PDP 11/70 33
4. Verification of the Synthetic Program's Workload 37
5. Correlation of Synthetic Response Time and Directory Transfers 39

1. INTRODUCTION

Measurement of computer systems performance has existed since the earliest days of computing, but the introduction of multiprogramming created the need for more comprehensive measurement of overall performance. The physical speeds of individual components are no longer sufficient to describe the speed of an entire system, since the overlapping of system resources is such a major factor. This report considers the performance of interactive computer systems, where many users can be simultaneously working at terminals.

The two main types of measurement for computer systems are throughput and turnaround time *. System response time is a special case of turnaround in interactive systems and is a predominant factor in assessing the level of performance of such systems. The speed of a system's reply to an action is important to users for its effect on individuals' workrates [10], and to management as an indication of the system's ability to fulfil its intended purpose. A quantitative measure of response time would be helpful to both groups. In the short term, programmers could decide if on-line work at a particular time would be productive or time-wasting. Managers could tell whether the system's performance met design or contractual goals, and take action if limits were exceeded [3]. Correlation of a measured response time against other performance measures

---

* Rosen [13] adds quality as a third type, e.g. the number of system crashes per week.

can help locate the causes of poor response time *.

The material covered in this report began as part of a study of performance at a timesharing computer bureau. CBL (Canterbury) Ltd. operates a PDP 11/70 to provide a timesharing service for customers in the Christchurch area. During the day commercial users, typically with enquiry systems accessing large databases, and CBL's own staff with program development compete for the machine's use. This study dealt with the service offered in office hours, ignoring the sometimes heavy work done at other times. In terms of this study, the performance of the machine outside normal hours was unimportant, provided that batch work did not interfere with the timesharing service.

The machine's performance was examined with the manufacturer's software monitor. A feature added to the routine which recorded data from the monitor was an early version of the response time measuring program described here. This proved useful in finding the times at which response was slow, enabling the distinguishing features of such periods to be found by examining the other statistics gathered. It must be emphasised that the measure described

- - - - - - - - - - -

* In this context, many sets of data must be collected to reduce the influence of random fluctuations in performance. An analysis of these random moment-to-moment changes, and the interpretation of the collected data are beyond the scope of this report. However it should be noted that the derivation of mathematically significant correlations between response time and other variables is not sufficient. The data extracted must be coupled with detailed knowledge of the operating system to produce plausible explanations and to suggest possible cures for poor performance.

here should be only part of a more comprehensive statistics collection scheme. Other measurements must be taken and decisions made with all factors in mind.

2. BACKGROUND

The overall performance of a system must often be stated in simple qualitative terms as good, bad or indifferent for management of the system. For example, at some point a decision has to be made that performance, especially response time, is poor enough to warrant committing valuable resources, such as personnel, to its improvement. There have been few if any attempts to construct measures to quantify an entire system's performance, because no single type of measurement can adequately describe a system's performance to the satisfaction of all concerned with its operation. Nevertheless some concise descriptions of system performance have been suggested, of which the Kiviat diagram is the best known [5]. Yet typically users of interactive systems are uninterested in the many specific measures available to systems programmers. There may be a great number of detailed statistics gathered, but users "...are interested only in the level of computer service delivered at the terminal" [1]. Thus the problem is to provide some measure which is closely related to the service which users are receiving.

2.1 Response Time

The one measurement especially important to timesharing users is system response time at terminals. Definitions of this have been attempted by several writers. Rosen defines this as being " ... measured from the time the user strikes a key that transmits a request for service (a carriage return or other attention key) to the time the response to that request starts printing at the terminal" [13].

Kelisky's definition, "System response time is the time measured from the user's signal to the system that there is work to be done to the point at which the system begins to present those results to the user" [10], is also from the user's viewpoint. Svobodova, who says, "The system response time consists of the time it takes to acknowledge a terminal request (reaction time) and the time needed to process such a request (CPU and I/O time)" [15], considers the work of the system in making the reply. Abrams and Treu [1] take the analysis of the system's internal working further by giving six components that make one interactive 'transaction'. Nevertheless they say that "A preferred definition is the elapsed time from the last keystroke made by the user until the first printing character is displayed at the user's terminal."

The uniformity of these definitions shows that if the elapsed time (as defined by Abrams and Treu) for any particular request can be adequately measured, it can confidently be said to be 'response time', though with qualifications relating to the nature of the request. Such time could be measured by hardware, with no record taken of the complexity of the request. This does not enable us to state the time in general terms as a system's response time, which in some way must be an aggregation of many actual or possible requests to a system. As Svobodova says, "one cannot expect the response time for very involved requests to be as short as the response time for trivial requests [16]". Hence Svobodova suggests that some percentile of response time is a better measure. A percentile has the disadvantage that its associated distribution involves recording many response times, and is a measure which applies to a longer interval than the brief time which one

interaction needs.

It appears that the conventional definition of system response time has two forms, with associated failings;
(1) The single interaction, which finds response time for one particular request at any one time. This is not a measure of performance over long intervals, and applies only to the one type of request made.
(2) The aggregation of many single measurements into distributions of response times for particular requests, which evens out the peculiarities of individual requests. This yields a figure which applies to some longer time period, in which the system's usage pattern may have been changing. In the long term this may change markedly, as during the transition from interactive to batch work at the end of a day.

2.2 Benchmarking

The measurement of the time for a single actual request is an approach similar to benchmarking, which has long been an accepted method of measuring computer performance [11], though mainly for comparisons between different installations or for choosing new systems [2]. One approach to benchmarking in interactive computing has been the use of 'scripts', which are actual loads imposed on the system from real or virtual (pseudo) terminals *. These have been mainly for investigating the number of terminals that can be supported for a given system response time (in the sense of the original definitions given earlier). This does not

- - - - - - - - - - -

* For example, DEC supply a User Environment Test Package (UETP) with RSTS/E, though primarily as "as automated means to verify the reliability of RSTS/E hardware".

measure the level of service to the users, but the capacity of the system to handle many users.

The basis of work done by a benchmark can be either real programs chosen from jobs normally run at the installation, and chosen to be representative of the workload, or synthetic programs which have no other purpose. The latter have the important advantage that they can change as the system's long term use changes. The best known synthetic job used for benchmarking is that of Buchholz [7], and variations of this have been used in several major comparisons of performance [2,14], Its attraction is that it can be given parameters to modify its use of resources, enabling it to be tuned to a particular system's characteristics. For example, Sreenivasan and Kleinmann [14] used spectral analysis in their careful matching of desired and actual characteristics. This paper suggests an approach similar to benchmarking, in that the elapsed time to run a job is taken as a measure of performance. The job itself is able to perform different mixtures of resource usage to reflect current usage of the system. It aims to compete for resources in the same ratio as currently running user jobs, so like Sreenivasan's work it is parameter driven, though its execution occurs much more immediately.

2.3 System Monitors

Collection of system statistics is by hardware or software monitors *. Software monitors, whether event or

- - - - - - - - - - -

* The term monitor is unfortunate, as it has other meanings in some descriptions of system software [6]. Rosen [13] uses 'probe' to describe the hardware device or software code that collects statistics, and this seems a worthwhile change of terminology. However this paper will use the more common though sometimes confusing name, 'monitor'.

timer driven, provide the more flexible tool, but leave most programmers dependent on the monitor's designers. Code for a monitor must be built into an operating system when it is written, and may well be inaccessible to applications programmers *. It is assumed in this study that the software monitor used is trustworthy; however it is recognised that a hardware comparison such as Peterson's [12] could reveal inaccuracies in the statistics measured by software.

One major disadvantage from which software (but not hardware) monitors suffer is that they themselves can use significant amounts of machine resources **. However all systems of the type considered here need some minimal form of statistics gathering which must be run during normal operation for accounting purposes, and this information could be used if no software monitor was available.

- - - - - - - - - - - - - -

* The statistics option for the RSTS/E system discussed later was a hidden feature until release 6C of RSTS/E. Even now DIGITAL do not support this monitor because "... we do not understand the numbers sufficiently to be able to explain them and their effect on performance" [M. Minow in correspondence to Professor J.P.Penny].

** The RSTS system's statistics option is stated to use between 1% [R.Marino, "Performance Evaluation", from his "A Byte Of Insight" series on technical aspects of RSTS/E, presented at the Chicago Spring DECUS meeting, April 1978] and 2% to 3% [M.Minow, documentation for STATS, the RSTS/E statistics collection program] of CPU time, which is a low enough figure to be acceptable during normal operation. This is not the case with all monitors. For example, the Purdue MACE software probe, which collects much more information on each job, uses 6% to 10% of available CPU time [17], and is not normally run.

3. A SUGGESTED RESPONSE TIME MEASURE

3.1 The Proposed Method

The proposed measure of response time is the elapsed time to complete the average actions of all current users. This time could be called the synthetic response time, belying its origins in the synthetic program described here. It has several characteristics which differentiate it from previous benchmarks. For example, one similar measure, the THI index at IBM's Yorktown Labs [10], used the elapsed time to run a sample job as an immediately available measure of system performance, but seems to have always run only a constant amount of work. The present sample job changes its own characteristics each time it runs to reflect the system's current usage, and runs as closely as possible to the time at which the controlling measurements were made.

As the users' requests change, so the nature of the synthetic program will change. There are at least two ways of doing this, and only one is closely examined here.
1. Decide beforehand which requests are worth considering. Count all such requests during some time period, then run one or more samples of all possible requests, recording the elapsed times. Calculate a composite response time, which is

$$\sum_{\text{all } i} n_i \; t_i \; W_i$$

where n_i are the number of requests of each type recorded in the preceeding time period.

t_i are the measured elapsed times of the sample(s) of type i. For increased confidence, at the expense of extra machine

time, this could be the average of several identical samples.

W_i are weighting coeffients set according to the relative importance of type i. For objectivity, each W_i should be constant, but could be varied because, for example, compilations are of lesser importance since they do not contribute to application user satisfaction.

2. Break all requests into common basic units, collate the number of each unit, and compose a sample job with the corresponding number of units, scaled to a reasonable size.

The prime failing of approach 1 is the large number of possible requests, and the variability of one request's workload. For example, accessing a file might be considered a basic request, but the work involved could depend very much on which record is being accessed, the size of the file and whether or not the file is contiguous, i.e. the file's characteristics. If the set of possible user demands on the system was small, and each type of request had an associated constant activity (as might occur in a system completely dedicated to some terminal service), the first approach might be valid. It was not considered to be so in this study's environment, a system with in-house programmers and external applications users working simultaneously.

The second approach was used, with some refinements. The common basic units in this study are the overall statistics gathered by the software monitor for the entire system, not on a per job basis, though preferably by classes of jobs. Each of these measurements should satisfy all of the following criteria.

(i) Cover some significant part of system operation, which is able to be accurately reproduced by the synthetic program. This excludes most interactive commands, since they invoke system software and cause amounts of work which can vary with time. For example, file directories may change in size, changing the amount of work done by any process using the directories.

(ii) Be able to be collected over some common time period, and be accessible to the synthetic program. For example, recording statistics on tape for later analysis is not a suitable method.

(iii) If possible, apply only to useful programs, disregarding system overheads which are not chargeable to any user. This filtering should also exclude the synthetic program itself, any other operating system activity and, in the context of the case study described later, probably program development work *. The synthetic job will itself generate overheads, simulating the overheads induced by users. In addition, it is likely that recognisably different classes of users, such as programmers or outside users, will have different patterns of machine usage, and the synthetic job should if possible correspond to some group. This concedes that the averaging of user requests to the system for inclusion in the synthetic job is better if it has some help in making that average. Another important reason is the "political" consideration - management may be trying to optimize performance for the end users, the paying customers, even at the expense of their own staff, so the measure should apply only to the selected group.

- - - - - - - - - - -

* This is equivalent to choosing the important requests to be considered in the first suggested approach.

Unfortunately this level of statistics collection may not be available on many monitors.

3.2 Resources Used by a Sample Program

All system resources should be considered for inclusion in the synthetic job, though only those which are usable by more than 1 user at a time are suitable. In this context a system's resources include only hardware devices, since software items can be regarded as demands on hardware, with differing frequencies and ratios of usage of the basic components of hardware. Hence a synthetic job which mimics the system's present pattern does not need to reproduce the performance of specific programs, only the usage of hardware.

Given that the synthetic program sees user jobs only as users of the machine's resources, and copies that usage, the ways in which it can exercise each of these resources must be examined.

3.2.1 CPU. This is perhaps the easiest to exercise, by looping in assembler or a higher-level language.There is a choice between using just one or a very few types of instruction, with the simple idea of wasting CPU time, or executing a wide range of instructions, to exercise many parts of the CPU's repetoire (as Buchholz did by summing values from an in-core table [9]). The commonest approach is to have some arbitrary mix of instructions, on the premise that any mixture is more realistic than none at all. Here the purpose of using the CPU in a synthetic program must be considered. If it is to use the CPU for a predetermined time, it doesn't matter how the device is used, only that the desired time be accurately expended.

The task is simpler if only one small group of instructions are used because of the problem of determining timings. In this case optimization of the exercising code can be a problem, and is probably the reason why Buchholz varied the type of CPU work. Care must be taken that the code is not optimized out of existence. For example, a compiler could change

```
for I:= 1 to 1000 do
  J:=0;
```

to

```
I:= 1000;
J:= 0;
```

which has the same result but is much faster, destroying any attempt to impose work on the system.

3.2.2 Discs. Disc work can be measured in several ways, such as by the access rate, data transfer rate or the number of access requests waiting for the disc. The time taken for disc accessing depends greatly on the distribution of the accessed blocks' physical positions on the disc, and attempts (if any) to optimize the head's movements, as these determine the head seek times (for moving head discs). To reproduce this work, records at different known locations on a disc can be accessed. The work of opening and closing files, and finding spare space on a disc for extending files, is probably too variable to be included here. On systems with more than one disc, the balance of usage between individual discs can be considered by treating each disc as a separate resource, providing of course that the use of each disc can be measured.

3.2.3 Main memory. Any program must use some space in main memory to be able to run, so all programs will at some

time request core. The important factor is the delay due to competition for memory. In the synthetic program a request could be made for a certain sized portion of memory, and the time to satisfy the request measured. The problem is that these times can vary widely in practice; spare memory might be available at any moment, giving zero time, or programs may need to be swapped out to make room, with consequent unplanned usage of secondary storage. In view of the practical difficulties associated with simulating demand for memory, this factor was omitted from this study. It is recognised that this is a serious failing of the present scheme, but a correct way to synthesize demand for main memory has not been found. This omission would be especially serious in virtual memory systems, where the problems are heightened because backing store accesses are intimately associated with the use of main memory.

3.2.4 Other Resources. Single tape drives and line printers are among other resources which cannot be shared between users. Each must be dedicated to just one job, so competition does not exist except in the long-term perspective of allocating devices to jobs. Thus they have been excluded from this study. Also excluded are factors such as differing speeds of main memory modules *, and delays in moving data between the central computer and the terminals. The latter point could be especially serious. It seems wrong to create a figure for on-line response without using the very device, an interactive terminal, which distinguishes this type of computing from all others. This factor could be incorporated by sending text from the

- - - - - - - - - - -

* On the University of Canterbury B6700 this can cause the same work to be done at varying speeds by the CPU.

synthetic program to some available terminal, through the same software which normally drives the terminal. Thus the operating system's terminal handler would also be used. The complications of using such a scheme meant that no account was taken of terminal-handling. It is claimed that the results obtained with the program are still useful, since they are not meant to be an exact measure, but only to compare the response of the system at different times.

3.3 Priorities

The synthetic program's priority must be set to the same as the group of users being examined, so that it competes equally with them. In a system with fixed priority levels this is straightforward. If more than one level of priority need to be examined at once then several versions of this program would be run together, with consequent increased, possibly excessive, resource usage by the multiple copies. With dynamically altering priorities, the system's method of calculating priorities must be considered. When only relative I/O and CPU usage determine a program's priority, the present synthetic job will be set by the system to a representative level. Should the elapsed times between requests be used in the calculation, the arbitrary time that this program blocks itself could mean that the assigned priority value does not correspond to any class of user. The solution might be for the synthetic program to set its own priority, based on the average priority of users.

4. AN IMPLEMENTATION OF THE PROPOSED MEASURE

4.1 The Computer System

The method given in section 3 has been implemented on a DEC PDP 11/70, which runs under the RSTS/E operating system. During the period of the study the system expanded from 2 to 3 RP06 discs (with 176 Mbytes each), from 192 to 256 K words of main memory and from about 20 to 40 terminals. Its work was partly program development, with programming in Basic-Plus, and partly actual on-line applications, such as terminal enquiry systems.

RSTS/E has a statistics gathering option which can be requested at system generation time [11] to create a software monitor in the system.The monitor increments counters at fixed locations in main memory during normal operation, and the difference in the contents of these counters over any time period gives values for the activity during that period. STATS, a program from DEC for recording the monitor's statistics from minute to minute, was used as the basis for collecting 5 measurements which determined the synthetic program's characteristics. A worked example of the equations which convert the statistics into parameters for the program is given in Appendix 3.

4.2 The Synthetic program

The synthetic program was designed to run once a minute. To avoid using too large a share of the machine's resources, the desired constant time was set at 2 seconds. Less then 2 seconds could mean that too small a sample was run. The time must be such that for 'normal' loadings several timeslices are needed to complete the CPU work, and

the disc work should involve several accesses of a number of the disc locations allocated. Three percent or less of the available CPU time was used, which added to the statistics monitor's time meant that about 5% of CPU time was taken to measure the system.

The most difficult part of the study was finding the combination of specific requests to the system to cause a known amount of work to to be done. Both CPU and disc accessing had to match the desired amounts. To examine this, the program was run with a variety of inputs to its equations, and the results are shown in Appendix 4. An important issue was deciding whether the CPU work should be done together in one group or as a series of shorter bursts, one before each disc access (i.e. interleaved with disc accesses). The experiment described in Appendix 4 was run in both modes, and the least squares correlation coefficients obtained were:-

(i) Interleaved:

measured CPU time = 0.89 requested time + 0.144 seconds

(ii) Non-interleaved:

measured CPU time = 0.97 requested time + 0.005 seconds

Hence the non-interleaved method was chosen.

4.3 The Resources Used

4.3.1 CPU. The CPU work in the synthetic program was a loop of the form,

A%=0% FOR I%=1% TO NUMB.CPU%

in RSTS/E's Basic-Plus *. This is a single operation performed many times, which avoids the optimizing problem only through the interpretive nature of Basic-Plus. The language interpreter, which translates and executes intermediate code, is invoked each time the loop is repeated.

The choice of Basic-Plus over assembler for the synthetic program is debatable. On the one hand it removes the program one step further from the basic machine exercising level at which it is designed to be. Alternatively, it offers the usual advantages of a higher level language, and is the predominant language at the particular installation examined.

Not surprisingly it was also found that the program, in calculating the sample's work and doing disc I/O, itself produced a measurable amount of CPU usage charged to the user. The CPU work requested was reduced by 8% of the requested I/O time to offset this.

4.3.2 Discs. Balancing the ratio of blocks to accesses for the disc work followed a scheme suggested by Professor J.P.Penny. Five files were opened, with record sizes of 1,2,4,8 and 16 blocks. An algorithm assigned transfers to

- - - - - - - - - - -

* The % signs indicate 16-bit integers, and NUMB.CPU% is the number of times that A% is zeroed.

files until the desired balance, or the closest balance within the fixed file sizes, was found. It may seem that including the data transfer times was not worthwhile because of the small times involved compared with average latency and seek times, but there were cases where disc accesses involved large numbers of blocks *. Our knowledge of the disc transfer mechanism is limited, so the assumption was made throughout this experiment that the disc is free to begin another I/O action as soon as the specified blocks have been moved. It is assumed that there is no need for the disc head to remain on the track until the current cluster (of 16 blocks on an RP06) is finished.

The 5 disc files were placed across the disc to ensure that disc head movements do occur, and the first record of a file was always specifically referenced. We have no measurements of the actual head movements, so the record's postions are rough guesses at locations which will approximate real disc usage **. The actual track and sector numbers are not known, but it is assumed that the retrieval block pointers values begin at one side of the disc and go in (or out) from there.

- - - - - - - - - - -

* This is especially true of swapping transfers, when entire programs are moved. Although swapping work was eliminated from the disc load input to the synthetic program's parameters, it does show that the calculated transfer times could be significant.

** The actual "retrieval block pointers" in accessing order are 8253, 40594, 8627, 32786 and 9735 on disk 0, and 10083, 12373, 29493, 10097 and 24503 on disk 1. The maximum possible pointer for RP06 discs is 41800. These are rather arbitrary positions determined by free positions on the discs.

The synthetic program must use the files one after another, that is, an access of one file must be followed, if possible, by a request to a different file to ensure that head movement does occur. One study [4] of disc accesses has shown that "the assumption of uniform distribution of disk addresses is not true". It is likely that this is also the case with RSTS/E where disc space is always allocated from address 1, meaning that unused blocks congregate at high disc addresses. Most of the work will be performed at one (inner or outer) part of the disc, and the average head movement will be less than that used in the earlier calculations. Hence the time spent in disc work is overestimated, moving the emphasis of the synthetic job towards disc accessing. Less CPU work is performed than is needed to reproduce the system's measured CPU/IO balance, and the sample job is more sensitive to disc activity. Since the average disc access time is longer for the synthetic job than for the system as a whole (because the synthetic job's files are widely spaced), the supposedly constant time for the synthetic job on an empty machine is distorted.

The area of disc access times remains a major problem in the development of a synthetic job. It would be better if the average disc head movement per disc access could be measured, enabling the synthetic program to reproduce this by accessing only some of a number of files at known uniformly spaced positions on the disc. However Boi et al [4] also showed that the distribution of accesses on sector positions was uniform, as might be expected, so at least the assumption that the average rotational delay is half the time for one complete rotation seems reasonable.

4.4 Results

The test of the project must be in the validation of the program's results. That is, how meaningful are the numbers it produces for response time. Unfortunately there is no clear way to do this.

One method would be to match the users' feelings on response time with the program's measures. Finding a reliable figure for user feelings is a difficult task. Perhaps grades of response time, from 1 (excellent) to 5 (very poor), could be collected at certain times of the day and consensus feelings found. Alternatively, users could note specific times when response was especially good or bad. Agreement between the times specified by users and those found by the synthetic program would be an important indication of success. The problem here is that the users must record their times as accurately as the program, which may be to the exact minute. Also the users' values are objective feelings, subject to change as the users become familiar with different levels of response. This particular aspect of validation was not done, and it would prove very difficult to do.

The most important indication of the suggested measure's validity is its practical usefulness. It is claimed that this has already been shown in the favourable results obtained with an early version of the synthetic program. As it was then, the program was inadequate in some of the aspects discussed earlier, such as performing the desired workload exactly as requested by the equations. Nevertheless its results correlated well with measurements of factors which limited system performance. Appendix 5

shows one such correlation, with disc transfers caused by accessing file directories plotted against measured response time. The system at that time had inadequate buffer space allocated to the directories, causing excessive disc traffic for directory blocks which slowed down the entire system.

5. SUMMARY

This paper has outlined a general method for the generation of a synthetic program, whose elapsed time to run is taken as a measure of the system's response. An implementation on a PDP 11/70 was given as an example. Design of such a program for any particular system needs access to a software monitor, and a comprehensive knowledge of the operating system. Aspects of implementing the program were discussed, but not its uses. However some examples were given to justify the claim that the measure is a valid indication of the system's user-related performance.

ACKNOWLEDGEMENTS

I would like to thank my supervisors Professor J.P.Penny and Dr. R.E.M.Cooper for their help with this project, and also CBL (Canterbury) Ltd. for the advice of their staff and the generous access to their machine.

REFERENCES

[1] Abrams, M.D., & Treu, S., "A Methodology for Interactive Computer Service Measurement", Communications of the A.C.M, vol 20, no 12, December 1977. pp 936-944.

[2] Barber, E.O., Asphjell, A., & Dispen, A. "Benchmark Construction", Performance Evaluation Review, vol 4, no 4, October 1975. pp 3-14.

[3] Bell, T.E., "Managing Computer Performance with Control Limits", Performance Evaluation Review, vol 4, no 1, January 1975. pp 21-28.

[4] Boi, L., Cros, P., Drucbert, J.P., & Rousselot, J.Y., "A Performance Evaluation of the CII SIRIS Operating System - Methodology, Tools and First Results", pp 67-80 in "Modelling and Performance Evaluation of Computer Systems", ed. E.Gelenbe, North-Holland, 1977.

[5] Borovits, I., & Ein-Dor, P., "Cost/Utilization: A Measure of System Performance", Communications of the A.C.M., vol 20, no 3, March 1977. pp 185-191.

[6] Brinch Hansen, P., "Operating System Principles", Prentice-Hall, 1973. p 336.

[7] Buchholz, W., "A Synthetic Job for Measuring System Performance", IBM Systems Journal, vol 8, no 4, 1969. pp 309-318.

[8] Digital Equipment Corporation, "RSTS/E System Generation Manual", DEC-11-ORGNA-B-D, 1977. Appendix E.

[9] Kelisky, R.P, "Managing Interactive Systems for User Effectiveness", pp 93-107 in "Interactive Systems", Springer-Verlag Lecture Notes in Computer Science, no 49.

[10] Lucas, H.C., "Performance Evaluation and Monitoring", Computing Surveys, vol 3, no 3, September 1971. pp 79-91.

[11] Peterson, T.G., "A Comparison of Software and Hardware Monitors", Performance Evaluation Review, vol 2, no 2, June 1974. pp 2-5.

[12] Rosen, S., "Lectures in the Measurement and Evaluation of the Performance of Computing Systems", S.I.A.M., Regional Conference Series in Applied Mathematics, #23. 1976.

[13] Sreenivasan, K., & Kleinmann, A.J., "On the Construction of a Representative Synthetic Workload", Communications of the A.C.M., vol 17, no 3, March 1974. pp 127-133.

[14] Svobodova, L., "Online System Performance Measurements with Software and Hybrid Monitors", Operating Systems Review, vol 7, no 4, October 1973. pp 45-53.

[15] Svobodova, L., chapter 2 in "Computer Performance Measurement and Evaluation Methods: Analysis and Applications", American Elsevier, 1976.

APPENDIX 1.

GENERAL EQUATIONS FOR THE SYNTHETIC PROGRAM

Suppose the sample program uses n resources for which the amounts of usage of each can be equivalenced to some common unit. For example, assuming (rather simple-mindedly) that disc accesses are evenly spread across the disc, the average disc access will take the sum of the average head seek time, the time for half a rotation of a disc, and the actual time spent transfering data. Similarly, a utilization percentage for the CPU can be converted in 'millisecs of use per second'.

Call each of these measured statistics, x , where each statistic applies to exactly the same time period. The aim is to construct a sample program which would take a constant elapsed time to run if it had the system all to itself, whatever its internal mixture of resource usage. Let this constant time be K seconds. Each of the resources' measurements must be converted from individual units to the common time units by the functions

(1) $$Y_i = f_i(X_i) \qquad i = 1..n$$

These functions can include weighting factors to reduce the importance of some measurements. Otherwise each resource which can be described by more than one type of measure will exert undue influence in the constuction of the sample.

The sum of these times is:

(2)

$$T = \sum_{i=1}^{n} Y_i$$

which enables a common ratio to be found

(3) $$R = K / T$$

which when multiplied by the known times will give the sample's desired time usage of each resource

(4) $$B_i = R \cdot Y_i$$

The B 's are converted to measureable (individual) units by the reverse step to equation (1);

(5) $$A_i = f_i^{-1}(B_i)$$

This determines the number of basic units to be performed by the program. The process of changing the A_i's into actual requests on the system depends very much on the actual system used, and no general method can be suggested. Section 4 of the report gives an example of tuning the synthetic program to perform the desired amount of basic (measurable) units.

APPENDIX 2

RESPONSE TIME MEASUREMENT PROGRAM FOR THE PDP 11/70

This appendix describes details of running the response time measuring program on a PDP 11/70 using RSTS/E.

The program listed here is designed to run by itself, collecting the necessary statistics and printing the measured response time to file #1. Only two discs can be sampled since each disc needs 5 files, and Basic-Plus allows only 12 files to be used.

The results can be matched with statistics gathered by STATRP (the modified STATS) by the following steps.

1. Run STATRP with a command string which includes the switches /D2 and /SL:60 . Do not use the /RESP switch, as that option uses and older version of response time measurement, on only one disc.
2. Start RUNRSP, with the same sleep time as STATRP was given. The privileged account used here must have access to both discs used.
3. After both programs have finished, run RSFIXR to rewrite STATRP's file with RUNRSP's response time on the last line of each minute's report.
4. STAN2 can then provide the desired reductions on the new file.

RUNRSP and STATRP cannot be incorporated into one program because the maximum program size of 16K would be exceeded. However it is possible to have these two programs communicating by messages, eliminating the need to ammend the statistics files after the programs are finished.

An example of STAN2's output is given in Appendix 5. This program extracts desired information from the recorded statistics and displays it in tabular or graphical form. The particular example shows data for 27 January 1978, from 9:55 to 11:40. STATRP recorded data every minute, which was later averaged into 5 minute intervals. The graph shows disc accesses involving file directory blocks on the horizontal axis, and response time vertically.

```
10      !
        !       R E S P O N
        !
20!     NAME:   RESPON
        AUTHOR: CHRIS SHEEDY
        DATE:   14-OCT-78

30!     DESCRIPTION:    THIS IS A STAND-ALONE VERSION OF THE
                RESPONSE TIME MEASURING SYNTHETIC PROGRAM.
                WATCH FOR LATER VERSIONS WHICH SHOULD BE LINKED
                TO E.G. STATRP BY MESSAGE SEND/RECEIVES
                TO GET AROUND THE 16K PROBLEM
        WARNING: THIS JOB DETACHES ITSELF!
                (BUT ONLY IF OUTPUT FILE IS NOT "KB:")

99      EXTEND          !ALLOW LONG VARIABLE NAMES

910     DIM T(100)
                !T()    OLD CPU DATA FOR INCREMENTAL STATISTICS

1005    DIM AV.X(2), AV.Y(2), XFER(2,5),
        L(6), P(2), Q(2)

1020    P7%=PEEK(PEEK(156%))\ U9%=PEEK(P7%) AND 255%
        \P8%=PEEK(PEEK(156%)+2%)
                !P7% -> DISK
                !P8% -> JOB
                !U9% := NUMBER OF DISKS IN CONFIGURATION

1030    IF (P7% OR P8%) = 0% THEN
                PRINT "?NO STATISTICS CONFIGURED ON THIS SYSTEM"
            \GOTO 32767
                !MUST HAVE JOB, DISK BUFFERS SYSGEN'D IN

1990    DATA "KB:", 60, 10
2000    READ DEFAULT1$, DEFAULT2%, DEFAULT3%
        !OUTFILE,       SLEEP,          NUMBER OF SLEEPS

2100    DISK.0% = 0%  \  DISK.1% = 1%
2200    PRINT "OUTPUT FILE <";DEFAULT1$;">";\ INPUT OUTFILE$
        \OUTFILE$ = DEFAULT1$ IF LEN(OUTFILE$) <= 0%
        \OPEN OUTFILE$ FOR OUTPUT AS FILE 1%
2240    PRINT "SLEEP <";DEFAULT2%;">";\ INPUT YAWN$
        \IF LEN(YAWN$)  THEN YAWN%=VAL(YAWN$)
                        ELSE YAWN%=DEFAULT2%
2250    PRINT "HOW MANY SLEEPS <";DEFAULT3%;">"; \ INPUT MORNINGS
        \ IF LEN(MORNINGS) THEN MORNING%=VAL(MORNINGS)
                           ELSE MORNING%=DEFAULT3%
```

```
2300    INPUT "INTERLEAVED";QS
        \IF LEFT(QS,1%) = "Y" THEN INTERLEAVED%=1%
                        ELSE INTERLEAVED%=0%
                        \PRINT "NOT INTERLEAVED"
                        \PRINT #1, "******  NOT ";
2310    PRINT #1, "INTERLEAVED DISK/CPU USAGE"
2320    PRINT #1, DATE$(0),TIME$(0)
2330    PRINT #1%, "DB0: XFERS", "DB0: BLOCKS", "DB1: XFERS", "DB1:BLOCKS",
                        "USERS' CPU %"
2340    PRINT #1%

2400    NUMB.DISKS% = 2%
        \ TD= 2000      ! MILLISECONDS DESIRED CONSTANT TIME FOR SAMPLE
2450    FIRST.TIME%=1%
        \IF LEFT(OUTFILES,3%) <> "KB:" THEN
                PRINT "DETACHING..."
                \QS = SYS(CHR$(6%) + CHR$(7%))

2500    FOR DAY%=1% TO MORNING%
2520    WAKEUP=TIME(0%)
        \GOTO 3450 IF FIRST.TIME%
2550    TRANS.5= FND(1%,DISK.0%,0%) + FND(1%,DISK.0%,2%)
        \TRANS.6=FND(1%,DISK.1%,0%) + FND(1%,DISK.1%,2%)
2560    BLOKS.5= FND(1%,DISK.0%,1%) + FND(1%,DISK.0%,3%)
        \BLOKS.6=FND(1%,DISK.1%,1%) + FND(1%,DISK.1%,3%)
3000    !

        !       C O L L E C T     J O B     S T A T I S T I C S

3200    L(0)=FNX(52%)           ! PR0
3210    L(1)=FNX(56%)           ! PR1
3220    L(2)=FNX(60%)           ! PR2
3230    L(3)=FNX(64%)           ! PR3
3240    L(4)=FNX(68%)           ! PR4
3250    L(5)=FNX(72%)           ! PR5
3270    U2=0
        \U2=U2+L(I%) FOR I%=0% TO 5%

3280    IF U2 > 0 THEN CPU.USER=(L(0)*100)/U2
                  ELSE CPU.USER=0
3400    AV.X(0),TRANS.3 = TRANS.5 - TRANS.1
        \AV.X(1),TRANS.4= TRANS.6 - TRANS.2
3410    AV.Y(0),BLOKS.3 = BLOKS.5 - BLOKS.1
        \AV.Y(1),BLOKS.4= BLOKS.6 - BLOKS.2
3420    GOSUB 26000
3430    GOSUB 27000
3440    GOSUB 28000
```

```
3450    TRANS.1= FND(1%,DISK.0%,0%) + FND(1%,DISK.0%,2%)
        \TRANS.2=FND(1%,DISK.1%,0%) + FND(1%,DISK.1%,2%)
3460    BLOKS.1= FND(1%,DISK.0%,1%) + FND(1%,DISK.0%,3%)
        \BLOKS.2=FND(1%,DISK.1%,1%) + FND(1%,DISK.1%,3%)
3550    TRANS.3 = TRANS.1 - TRANS.5
        \TRANS.4= TRANS.2 - TRANS.6
3560    BLOKS.3 = BLOKS.1 - BLOKS.5
        \BLOKS.4= BLOKS.2 - BLOKS.6
3980    T2$ = LEFT(TIME$(0%),5%) + ":" +
                MID(NUM1$(160%-(PEEK(516%) AND 255% )),2%,2%)
3990    PRINT #1%, DATE$(0) + "  " + T2$;
        TAB(25); T0; TAB(40); "RESPONSE TIME"
4000    PRINT #1%, AV.X(0%), AV.Y(0%), AV.X(1%), AV.Y(1%), CPU.USER

4010    PRINT #1%, P(0%), Q(0%), P(1%), Q(1%)

4020    PRINT #1%, TRANS.3, BLOKS.3, TRANS.4, BLOKS.4

4040    PRINT #1%, "TD=";TD, "T0=";T0, "TD.CPU=";TD.CPU, "T1=";T1

4050    PRINT #1%

4100    PILLOW%=YAWN-(TIME(0%)-WAKEUP)
        \PILLOW%=1% IF PILLOW%<=0%
        \SLEEP PILLOW%
4120    FIRST.TIME%=0%
        \NEXT DAY%

4200    CLOSE #1%
4500    GOTO 32767
```

```
20090   DEF FNP (Q%)
        \Q=PEEK(Q%+P%)\ Q=Q+65536.0 IF Q < 0.0
        \FNP=Q+(65536.0*PEEK(Q%+P%+2%))
        \FNEND          !GET FLOATING TABLE ENTRY

20100   DEF FNX(Q%)
        \P%=P8%
        \Q = FNP(Q%)\ T(Q%) = 0.0 IF I9%\ FNX = Q - T(Q%)\ T(Q%) = Q
        \FNEND                  !INCREMENTAL TABLE VALUE

22000   DEF FND(C%,U%,I%)
        \P%=P7%+2%
        \FND = FNP ((C%*U9% + U%)*16% + I%*4%)
        \FNEND
                !GET DISK BUFFER DATA:
                !C%     WHICH (0-5)
                !I%     WHAT (0-3)
                !U%     WHERE (0-MAX UNITS)

22010   !DISK DATA IS STORED AS FOLLOWS
        !       C%              I% (ALREADY DESCRIBED)
        !       0 SWAPS AND RTS READS
        !       1 USER I/O
        !       2 SAT
        !       3 OVERLAY CODE
        !       4 DIRECTORY I/O
        !       5 DECTAPE BUFFER
```

```
26000!  ************************************************
        *                                              *
        *       C A L C U L A T E                      *
        *       P A R A M E T E R S                    *
        *       FOR RESPONSE-TIME SAMPLE.              *
        *                                              *
        ************************************************

        GIVEN
        -----
        TD              (MS) DESIRED CONSTANT 'MACHINE TIME' FOR SAMPLE
        CPU.USER                (% UTILISATION) CPU TIME FOR USERS' PROGRAMS
        AV.X(I%)        AVERAGE XFERS PER SECOND, FOR DISK I%
        AV.Y(I%)        AVERAGE BLOCKS PER SECOND, DISK I%
        NUMB.DISKS%     NUMBER OF DISKS BEING USED IN SAMPLE

26010   !

        USES
        ----
        I%              GENERAL PURPOSE COUNTER
        T.CPU           (MS/SEC) CPU TIME (MEASURED)
        T.IO            (MS/SEC) IO TIME (MEASURED)
        T.TOT           TOTAL 'MACHINE TIME', (MS/SEC)
                        = T.CPU + T.IO , SO HAS MAX OF 2
        T.I(I%)         T.IO'S  FOR INDIVIDUAL DISKS
        T.I             TEMPORARY FOR T.I(I%)
        TD.IO           DESIRED TOTAL IO TIME FOR SAMPLE
        R.I             RATIO OF BLOCKS/XFER FOR A DISK
        P               TEMP FOR P(I%)
        TD.I            DESIRED T.IO FOR AN INDIVIDUAL DISK

26020   !

        RETURNS
        -------
        TD.CPU          (MS) TIME FOR SAMPLE'S CPU WORK
        P(DISK%)                TOTAL TRANSFERS FOR SAMPLE, DISK DISK%
        Q(DISK%)                TOTAL BLOCKS FOR SAMPLE ON DISK DISK%


26050   GUTO 26100 IF CPU.USER>0.0
        \ TD.CPU=0.0
        \ FOR DISK%=0% TO NUMB.DISKS%-1%
                \GUTO 26100 IF AV.X(DISK%)>0
                \P(DISK%)=0.0
                \Q(DISK%)=0.0
        \NEXT DISK%
        \RETURN

26100   T.CPU= 10.0 * CPU.USER
        \ T.IO= 0.0
```

```
26120   FOR DISK%=0% TO NUMB.DISKS%-1%
                \T.I= (36*AV.X(DISK%)) + (0.767*AV.Y(DISK%)) - 0.122
                \T.I=0.0 IF T.I<=0.0
                \T.I=T.I/NUMB.DISKS%
                \T.IO= T.IO + T.I
                \T.I(DISK%)= T.I
        \NEXT DISK%

26140   T.TOT= T.CPU + T.IO
        \ T.TOT=0.01 IF T.TOT<=0
        \ TD.IO = (T.IO/T.TOT) * TD
        \ TD.CPU = TD - (TD.IO*1.06)
        \ TD.CPU = 0.0  IF TD.CPU<0.0

26160   FOR DISK%=0% TO NUMB.DISKS%-1%
                \T.I= T.I(DISK%)
                \IF ( AV.X(DISK%)>0.0 ) AND ( T.I > 0.0 ) GOTO 26180
                ELSE P(DISK%)=0.0
                \     Q(DISK%)=0.0
                \     GOTO 26190
26180           TD.I= (TD.IO/T.IO) * T.I / NUMB.DISKS%
                \P= (TD.I/T.I) * AV.X(DISK%)
                \R.I= AV.Y(DISK%) / AV.X(DISK%)
                \Q(DISK%)= R.I * P
                \P(DISK%)= P
                \TD.I(DISK%)= TD.I
26190   NEXT DISK%

26199   RETURN
        !**************************
```

```
27000! ********************************************************************
       *                                                                  *
       *       SUBROUTINE TO ADJUST PARAMETERS TO PERFORMABLE TYPE        *
       *                                                                  *
       ********************************************************************

       GIVEN
       -----
       TD.CPU          (MS) TIME IN SAMPLE'S CPU LOOP
       P(I%)           TOTAL TRANSFERS IN SAMPLE FOR DISK I%
       Q(I%)           TOTAL BLOCKS IN SAMPLE FOR DISK I%
       NUMB.DISKS%     NUMBER OF DISKS USED IN SAMPLE

       USES
       ----
       I%              GENERAL PURPOSE COUNTER
       J%              GENERAL PURPOSE COUNTER
       DIFF%           DIFFERENCE BETWEEN REQUIRED AND CALCULATED
                             NUMBER OF TRANSFERS
       DISK%           COUNTER OF DISKS

       RETURNS
       -------
       XFER(I%,DISK%)  NUMBER OF TRANSFERS OF SIZE 2^I% BLOCKS ON DISK%

27020  FOR DISK%=0% TO NUMB.DISKS%-1%
              \J%=INT(Q(DISK%)+0.5)
              \XFER%(DISK%,4%)= (J% AND 32752%) / 16%
              \XFER%(DISK%,I%)= (J% AND (2%^I%)) / (2%^I%)
                                                  FOR I%=0% TO 3%

27030  DIFF% = 0%
       \ DIFF% = DIFF% + XFER%(DISK%,J%)          FOR J%=0% TO 4%
       \ DIFF% = P(DISK%) - DIFF%
       \ GOTO 27199 IF DIFF% <=0%
27050  FOR J%=4% TO 1% STEP -1%
       \CHANGES%=0%
       \IF XFER%(DISK%,J%) THEN XFER%(DISK%,J%)=XFER%(DISK%,J%)-1%
                       \XFER%(DISK%,J%-1%) = XFER%(DISK%,J%-1%) + 2%
                       \DIFF% = DIFF% - 1%
                       \CHANGES%=1%
                       \IF DIFF% <= 0% THEN GOTO 27199
27060  NEXT J%
              \ GOTO 27050 IF CHANGES%


27199  NEXT DISK%
       \ RETURN
       !*************************
```

```
28000!  ****************************************
        *                                      *
        *       DO THE RESPONSE MIX            *
        *                                      *
        ****************************************


        GIVEN
        -----
        INTERLEAVED% =  0       DO CPU, I/O WORK SEPARATELY
                        1       DO THE WORK INTERLEAVED (I.E. ONE CPU BURST
                                        FOR EVERY DISK XFER)
        NUMB.DISKS%             NUMBER OF DISKS USED IN SAMPLE
        TD.CPU                  (MS) TIME WANTED IN CPU LOOP
        XFER%(DISK%,J%)         NUMBER OF XFERS OF SIZE (2%^J%) BLOCKS ON DISK%

        USES
        ----
        I%              GENERAL PURPOSE COUNTER
        I.3%            = 1% + 3%
        HERE.B4%        BOOLEAN, TRUE IF FILES ALREADY OPEN
                                (ASSUMED ORIGINALLY ZERO)
        NOT.FIN%        BOOLEAN, TRUE WHEN ALL DISK WORK DONE
        J%              GENERAL PURPOSE COUNTER
        V%              GENERAL PURPOSE COUNTER
        DISK%           COUNTER OF DISKS
        TOTAL.XFER%     TOTAL NUMBER OF TRANSFERS IN SAMPLE
        DELTA.CPU%      (MS*CONSTANT) SIZE OF CPU LOOPS AT EACH XFER,
                                IF INTERLEAVING
        BIGFILE%        INDEX OF CURRENT DISK'S LARGE FILE
        F$              FILE NAMES

        RETURNS
        -------
        T0      (SECS) ELAPSED TIME USED
        T1      (1/10 SECS) CPU TIME USED

28005   GOTO 28020 IF HERE.B4%
        \HERE.B4%=1%

28008   FOR DISK%=0% TO NUMB.DISKS%-1%
        \ V% = DISK%*5% \ L$ = "DB"+NUM1$(DISK%)+":RSTEST.LS"
        \CLOSE #I%+V% FOR I%=3% TO 7%
        \FOR I%= 3% TO 6%
                \OPEN L$+NUM1$(I%) AS FILE I%+V%, RECORDSIZE 2%
        \NEXT I%
28010   IF DISK% THEN OPEN L$+"7" AS FILE 7%+V%, RECORDSIZE    2
                 ELSE OPEN L$+"7" AS FILE 7%,    RECORDSIZE 8192

28015   NEXT DISK%

28020   TOTAL.XFER%= 0%
        \TOTAL.XFER%= TOTAL.XFER% + XFER%(DISK%,I%)
                                FOR DISK%= 0% TO NUMB.DISKS%-1%
                                FOR I%= 0% TO 4%

28040   T0= TIME(0%)    \ T1= TIME(1%)
        \ BIGFILE% = 7%         ! USE ONLY THIS ONE LARGE BUFFER
```

```
28060   IF (INTERLEAVED%=0%) OR (TOTAL.XFER%<=0%) THEN
                A%=0% FOR V%=1% TO  (TD.CPU*10.0)
                \GOTO 28180

28070   IF INTERLEAVED% THEN DELTA.CPU% = (TD.CPU*10.0) / TOTAL.XFER%
                        ELSE DELTA.CPU% = 0%
28080   FOR DISK%=0% TO NUMB.DISKS%-1%

28100           NOT.FIN%=0%
                \FOR I%=0% TO 3%
                        \ I.3%= I%+(DISK%*5%)+3%
                        \GOTO 28140 IF XFER%(DISK%,I%)<=0%
                        \NOT.FIN%=1%
                        \XFER%(DISK%,I%)=XFER%(DISK%,I%)-1%
                        \A%=0% FOR J%=1% TO DELTA.CPU%
                        \BLOCK.LENGTH%= 512% * (2%^(I%))
                        \PUT #SWAP%(BIGFILE%) + I.3%,
                                RECORD 1, COUNT BLOCK.LENGTH%

28140           NEXT I%
                \GOTO 28160 IF XFER%(DISK%,4%)<=0%
                \NOT.FIN%=1%
                \XFER%(DISK%,4%)=XFER%(DISK%,4%)-1%
                \A%=0% FOR V%=1% TO DELTA.CPU%
                \IF DISK% THEN PUT #SWAP%(BIGFILE%) + (7%+DISK%*5%),
                        RECORD 1, COUNT 8192
                ELSE PUT #BIGFILE%, RECORD 1, COUNT 8192%

28160           IF NOT.FIN% GOTO 28100
28170   NEXT DISK%
28180   T0= TIME(0%) - T0
        \T1=TIME(1%) - T1

28199   RETURN
        !***************************

30000 STOP
32767 END
```

APPENDIX 3

EXAMPLES OF EQUATIONS FOR PDP 11/70

The 5 statistics used to drive the synthetic program are:

1. X_1 , user CPU time, as a fraction of total elapsed time. At each clock tick at the line frequency of 50 Hz the monitor records the CPU's internal priority level, where user programs, in general, occupy level zero. The ratio of ticks in priority 0 to total ticks gives the measured user CPU usage.
2. X_2 , X_4 , disc accesses for two discs by user programs. At each request for disc I/O the monitor adds to running totals of user and of all disc accesses.
3. X_3 , X_5 , the number of blocks of 512 bytes involved in user disc I/O, recorded for the X_2 and X_4 accesses.

The functions to convert these measurements to time, as shown in appendix 1, are:

(i) $y_1 = 10\ X_1$

(ii) $y_2 = 36\ X_2$

(iii) $y_3 = 0.767\ X_3 - 0.122$

(iv) $y_4 = 36\ X_4$

(v) $y_5 = 0.767\ X_5 - 0.122$

where all units are in milliseconds. All functions are linear, which simplifies the reversing step to obtain the A_i 's of equation 5 in Appendix 1. The times for individual discs were summed (i.e. $y_2 + y_3$ and $y_4 + y_5$) and the times for each disc used as one value in calculating the ratios of

IO to CPU usage #. To reduce the importance of disc work in the sample, all disc times were divided by the number of discs. Thus T.IO (the sum of the disc times) and T.CPU (i.e. y_1) were of equal importance.

Suppose the measured values are

X_1 = 50% User CPU Utilization
X_2 = 5 User accesses of disc 0
X_3 = 20 Blocks moved to and from disc 0 by X_2 accesses
X_4 = 30 User accesses of disc 1
X_5 = 60 Blocks moved to and from disc 1 by X_4 accesses

The conversions to time are

$y_1 = X_1 * 10 = 500$ milliseconds
$y_2 = X_2 * 36 = 180$ milliseconds
$y_3 = X_3 * 0.767 - 0.122 = 15.2$ milliseconds
$y_4 = X_4 * 36 = 1080$ milliseconds
$y_5 = X_5 * 0.767 - 0.122 = 45.9$ milliseconds

T.CPU = y_1 = 500 milliseconds
T.Disc(0) = (y_2 + y_3) / Numb.discs
= 97.6 milliseconds
T.Disc(1) = (y_4 + y_5) / Numb.discs
= 563.0 milliseconds
T.IO = T.Disc(0) + T.Disc(1)
= 660.6 milliseconds
T.Total = T.CPU + T.IO
= 1160.6 milliseconds

Note that the times for the discs have been divided by the total number of discs (Numb.discs) to reduce the

- - - - - - - - - - -

The times for (iv) and (v) are the times to traverse 1 block and 1 inter-block gap, corrected by subtracting the time for 1 gap, since the gaps are 1 fewer than the blocks.

influence of IO on the synthetic program.

T.Desired.IO = (T.IO / T.Total) * T.Desired

= (660.6 / 1160.6) * 2000 milliseconds

= 1138 milliseconds

A_1 = T.Desired.CPU = T.Desired - (T.Desired.IO * 1.08)

= 771 milliseconds

The desired time for the CPU content of the sample includes a correction factor of 1.08 to allow for the time used in controlling the the disc work.

The times for the individual discs can now be calculated.

T.Desired.Disc(0) = (T.Desired.IO / T.IO) * T.Disc(0)

= (1138 / 660.6) * 97.6

= 168

A_2 = (T.Desired.Disc(0) / T.Disc(0)) * x_2

= 8.6

= 9 disc accesses

A_3 = (T.Desired.Disc(0) / T.Disc(0)) * x_3

= 34.43

= 34

T.Desired.Disc(1) = (T.Desired.IO / T.IO) * T.Disc(1)

= (1138 / 660.6) * 563.0

= 970

A_4 = (T.Desired.Disc(1) / T.Disc(1)) * x_4

= 51.7

= 52 disc accesses

A_5 = (T.Desired.Disc(1) / T.Disc(1)) * x_5

= 103.4

= 103 blocks

Thus the units to be done in the synthetic program are

A_1	=	771	milliseconds
A_2	=	8	disc accesses
A_3	=	34	blocks
A_4	=	52	disc accesses
A_5	=	103	blocks

APPENDIX 4

VERIFICATION OF THE SYNTHETIC PROGRAM'S WORKLOAD

The program was run with the following input to its set of equations.
CPU utilization by users: 20%, 50% and 80% (3 values)
User disc blocks per second and transfers per second, on each of two discs:

4 (blocks) / 2 (transfers), 20/2, 20/10, 100/10, 40/20, 200/20 (6 values)

Each of the 18 combinations was repeated 6 times, giving 108 runs of the program.

The CPU time used was measured with a system intrinsic, TIME(1), which returns the total CPU time in tenths of seconds for the program.

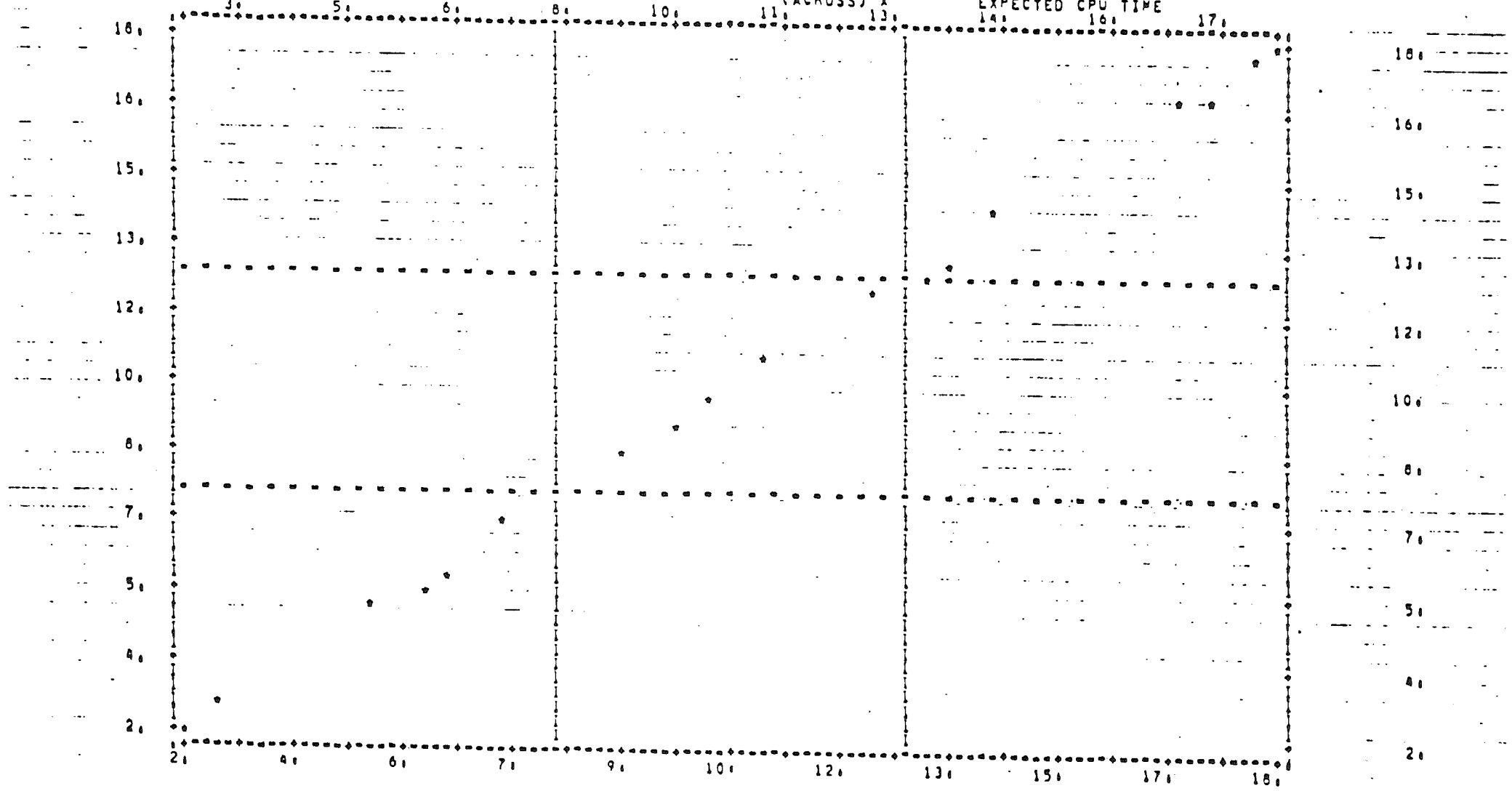
Disc work for the program can be measured only by running the program when the machine is otherwise completely idle. At such times all measured disc work for the entire system is attributable to the synthetic program.

The accompanying SPSS output shows requested CPU work in tenths of seconds on the horizontal axis and the measured time vertically. Each point on the graph is the average of 6 trials.

The least squares line is

$$y = 0.97\,x + 0.005 \quad \text{seconds}$$

FILE NONAME (CREATION DATE = 10/24/78)
SCATTERGRAM OF (DOWN) Y (ACROSS) X

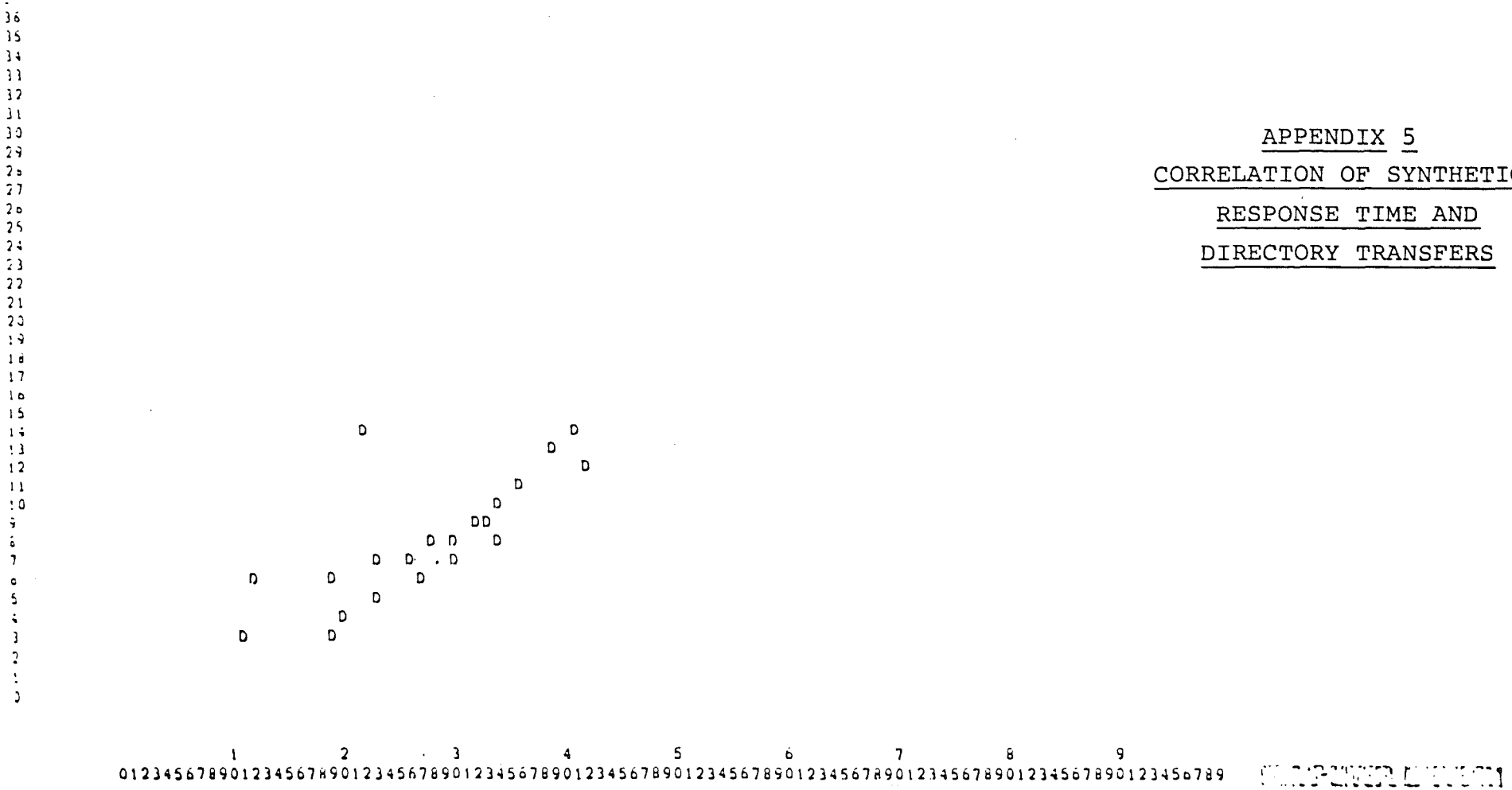


STATISTICAL ANALYSIS FOR 27-JAN-78 STARTING AT 06:01:32
START TIME = 09:55:00 FINISH TIME = 11:40:00 STATISTICS ARE AVERAGED OVER 5 MINUTE INTERVALS

CORRELATION DIAGRAM 4

PAIR	SYMBOL		HORIZONTAL	VERTICAL	RANGE MIN	RANGE MAX	GROUPS	BELOW RANGE	ABOVE RANGE
1	D	222	BOTH DIRECTORY XFERS/SEC		0	100	1	0	0
		1	RESPONSE TIME		0	40	1	0	0

NUMBER OF POINTS = 21



APPENDIX 5

CORRELATION OF SYNTHETIC RESPONSE TIME AND DIRECTORY TRANSFERS