A hybrid computer performance modelling system

Eric Foxley

Department of Mathematics, University of Nottingham, University Park, Nottingham NG7 2RD

The paper describes a method for modelling computer performance using a combination of analytic modelling and discrete event simulation techniques. The hybrid method is able to represent a variety of workloads and scheduling/deadline systems better than either technique separately.

(Received November 1976)

1. Introduction

Most modelling of computer systems falls into one of the following two categories:

(*a*) analytic models (usually based on queueing theory), in which a mathematical model of the system is formulated. This model assumes a knowledge of the hardware configuration and the statistical distribution of resource demands by the programs running, and is solved analytically. Typical examples of this approach are Berners-Lee (1972), Buzen (1971) and Chen (1975). Other more pragmatic approaches are possible, such as fitting a function to a known performance value.

(*b*) discrete event simulation, in which the detailed operation of the whole system is simulated at a low level, using as discrete events program interrupts, and peripheral responses, for example. A typical example of this approach was IBM's research work for the design of the 360/85 câche memory system.

The major advantages and disadvantages of the two approaches include

1. *Analytic Models*

The main advantage of these models is the small amount of computation involved in their solution. Once the model has been determined analytically, specific solutions are readily computed. The main drawbacks relate to the statistical assumptions that must be made for the model to be solvable analytically. These assumptions usually include, for example, equilibrium conditions and exponential distribution of resource demands. The equilibrium condition is particularly unrealistic, since the loading of a large core job in a multiprogramming system can suddenly reduce the number of active jobs significantly. The requirement for exponential distribution of resource demands is now being relaxed in some models (Gaver, 1971; Kobyashi, 1974; Salman, 1976). Another feature of real computer systems which is only just beginning to enter analytic modelling in a restricted form is that of job priorities and deadlines (Chow, 1975; Mitrani and Hine, 1975; Coffman and Mitrani, 1975).

2. *Discrete event simulations*

Such simulations usually run slowly, orders of magnitude slower than real time, and so are expensive in their computation demands. However, they do allow a detailed study of particular hardware features to be made, and are valuable at the hardware design stage. In addition, time dependencies can be built in, so that nonequilibrium conditions can be simulated.

It is proposed that, to estimate the performance of a general purpose computer system whose workload includes a large variety of jobs, an intermediate method is most likely to offer the best results in terms of a balance between accuracy and computation required. The following method has been designed to fulfil this need, and yet allow the analytic/discrete event interface to be moved up or down to obtain optimum results. The area of 'hybrid' simulation has been little explored, the only known reference being Kimbleton (1974).

The hybrid system to be described involves the simulation of a workload queue (of jobs with particular statistical distributions of resource demands) and a programmed job scheduler (which starts jobs and swaps them in and out if required, thus defining a current active job mix from the jobs currently in the queue). After each entry to the job scheduler, an analytic model is applied to the current active job mix, to determine the speed of processing of each active job, and the time pattern is advanced to the next event using discrete event simulation techniques. Thus a situation in which a single large job has a pathological effect on the system is passed to the analytic model as appropriate and becomes included in the overall performance calculation in a realistic way.

A second reason for the choice of a simulation system involving a job scheduling algorithm arose from a previous involvement in defining a benchmarking system. The user

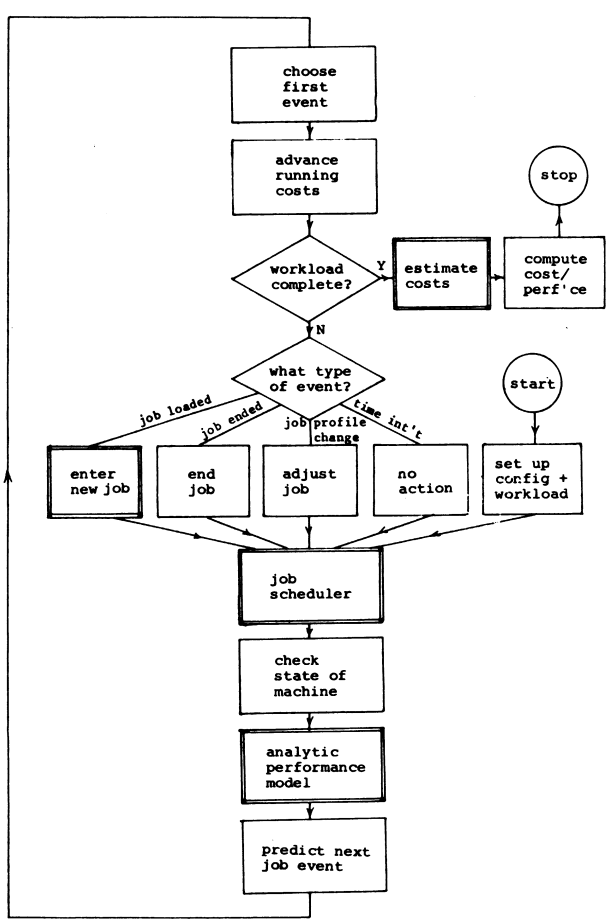

Fig. 1 Flow diagram of simulation process

view of the performance of a computing system involves not only the performance in terms of total throughput, but the ability of the system to meet deadlines and provide a good turnaround time. It is therefore essential to include in any performance calculations some measurement of the system's success in meeting deadlines and one must also define a job scheduler which makes decisions to balance a possible loss of throughput against a requirement to meet deadlines.

A third reason for implementing this system relates to its use in a university undergraduate operating systems class. It was felt that although one could teach, for example, core and processor scheduling the students never obtained a 'feel' for the scheduling problems in the provision of a general purpose computer service or for the balance between processor power and core size for optimum performance. A version of the performance measuring system has been built in which the student must supply a hardware configuration and a job scheduler to meet certain workload and cost requirements.

2. Overview

A flowchart of the modelling system is shown in **Fig. 1.** Each item in a double lined box represents a module which can be varied to represent different workloads/computer systems/job schedulers, etc. and is discussed further in the next section.

The flowchart represents essentially a simple discrete event simulation system. After each entry to the job scheduler, the state of the simulated machine is checked (for example, that the swap area has not been exceeded), and the analytic model is applied to the mix of jobs currently running. From this we determine the rate of processing of each active job, and can predict the time of the next job event (a job ending or changing its core demand, for example). This is then compared with other events (such as jobs being loaded, or time interrupts) and the first event determined and executed.

The cycle then repeats, and is terminated when the requisite number of jobs have been completed.

3. The Discrete event/analytic interface

A factor which offers great flexibility in this hybrid method of performance estimation is that the discrete event/analytic interface can be varied to provide different levels of analytic modelling without any fundamental change to the system.

For example, the job scheduler currently being used is in full control of swapping jobs to and from the drum, and the simulator checks that the core size and swap area are not exceeded. This enables a simple queueing model to be used. However, the interface could easily be moved in a more analytic direction by allowing the (effect of the) swapping to be included in the analytic model. The scheduler could now schedule more than the actual available store and the analytic model would have to include, for example, some kind of degradation factor to represent the swapping overheads incurred when the actual store is exceeded. Such analytic models for virtual store have been developed by Belady (1967), Brandwajn (1974), Courtois (1971) and Saltzer (1974).

4. System dependent modules

The four modules indicated with double lines in the flowchart of Fig. 1 represent between them the system being modelled. They will now be discussed in more detail.

4.1 *The analytic model*

Many models have been developed and published in this area. We require a model which, given the current mix of jobs, determines the rate of processing of each job. Most models can be easily amended to give this information.

The analytic model used in the system for teaching undergraduates is biased so that the performance degradation due

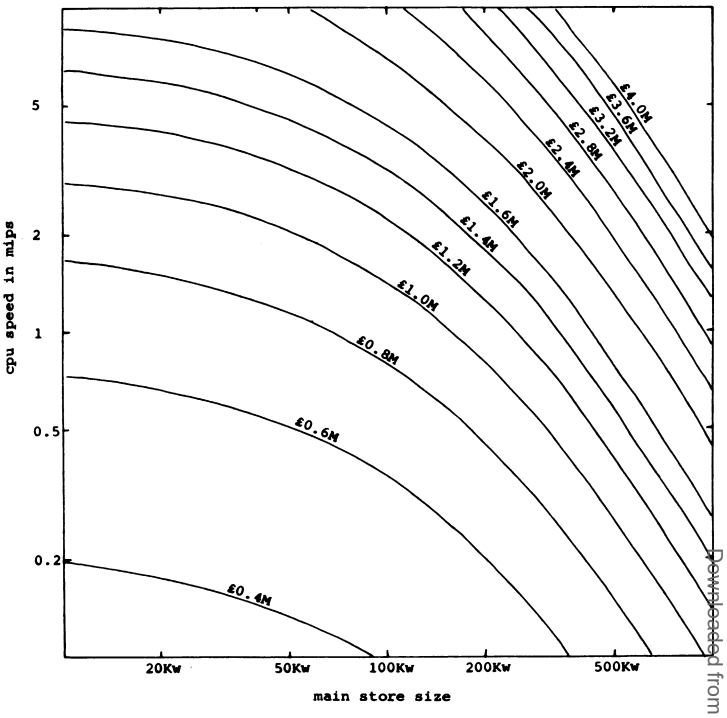

Fig. 2 Estimated configuration costs

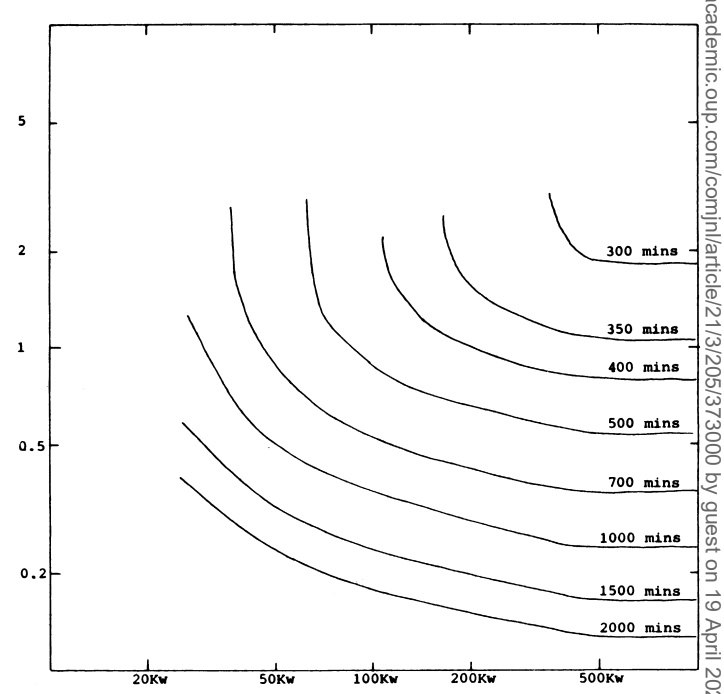

Fig. 3 Time to complete 100 jobs of simple workload

to a CPU/transput balance is exaggerated. This encourages the students to write schedulers which carefully maintain a balanced job mix.

4.2 *The simulated workload*

Each job in the simulated workload has a 'user specification' which can be inspected by the scheduler. This consists of a required completion time and 'user' estimates of elapsed time, core use and I/O bias (ratio of I/O activity to total activity). The job then has a runtime profile, taking it through a number of phases, during each of which its profile (core and I/O bias) is constant. Some of these phases may violate the user specification. The scheduler can inspect the instantaneous state of a

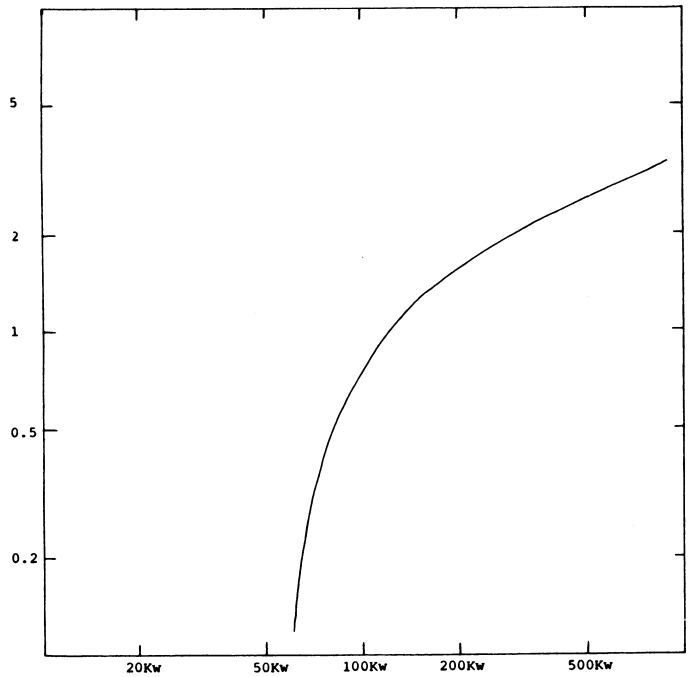

Fig. 4 Optimum configurations for simple workload

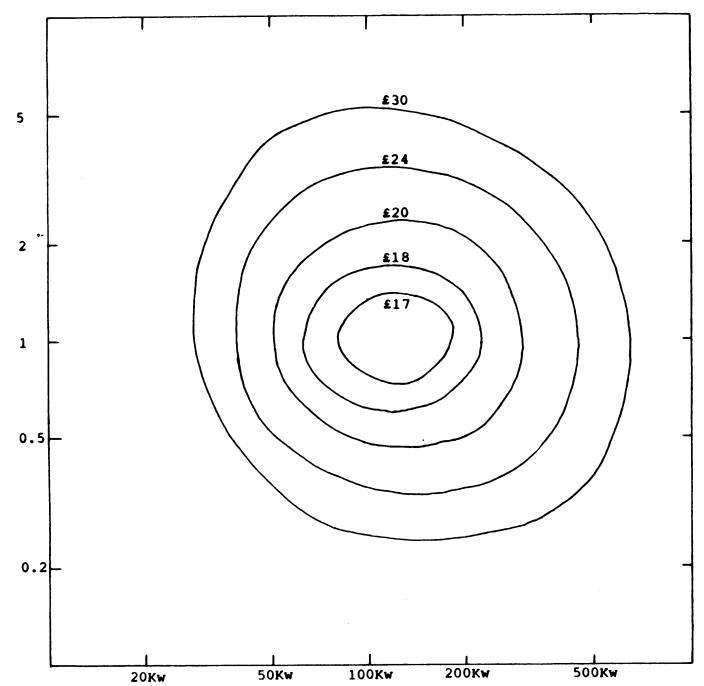

Fig. 5 Estimated cost per job for simple workload

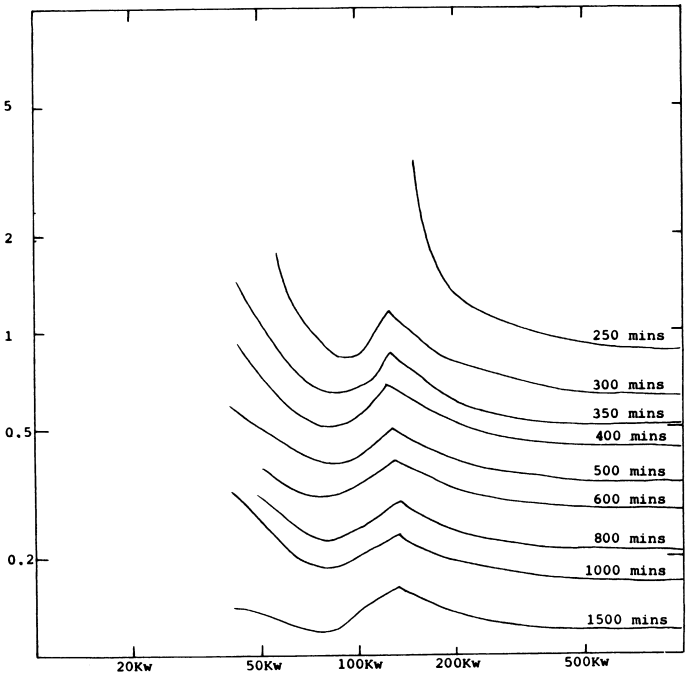

Fig. 6 Time to complete 100 jobs of University workload

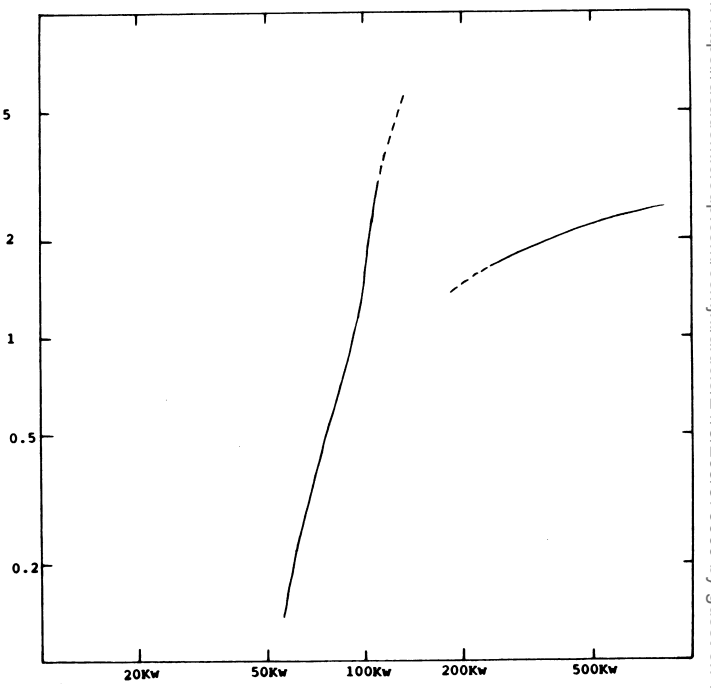

Fig. 7 Optimum configurations for University workload

job but cannot obtain information in advance of its runtime profile.

The workloads built into the system at present include two examples specifically for student use (the first a simple random workload, the second a 'real time' workload of small I/O bound high priority transactions plus background, all with accurate user specifications) and a number of workloads based on the local University Computing Centre's workload statistics. Because of the way the latter have been collected, they include some of the operating system overheads (such as directory accesses and accounting). If it was required to simulate the workload, as it would run on a different range of machines, variations in operating system and compiler overheads would have to be taken into account.

The simulation of timesharing workloads could easily be included, but the current examples all relate mainly to batch work.

4.3 *Hardware costs*

The formula deducing cost from the configuration specified is an ad hoc one, devised mainly with a view to student use, but based roughly on ICL 1900 series prices.

4.4 *The job scheduler*

There are obviously many objectives which may conflict with each other in the design of a scheduler. Should one maximise throughput (maximise core use, or optimise CPU/transput balance), or minimise the number of late jobs? Should one abandon jobs which violate their user specification? The balance between all these features depends on the type of system being modelled but all realistic systems can be modelled.

The penalties in throughput related to returning jobs on time are particularly complex. If the overall throughput is reduced too far average turnarounds must deteriorate and this is particularly significant on small store systems.

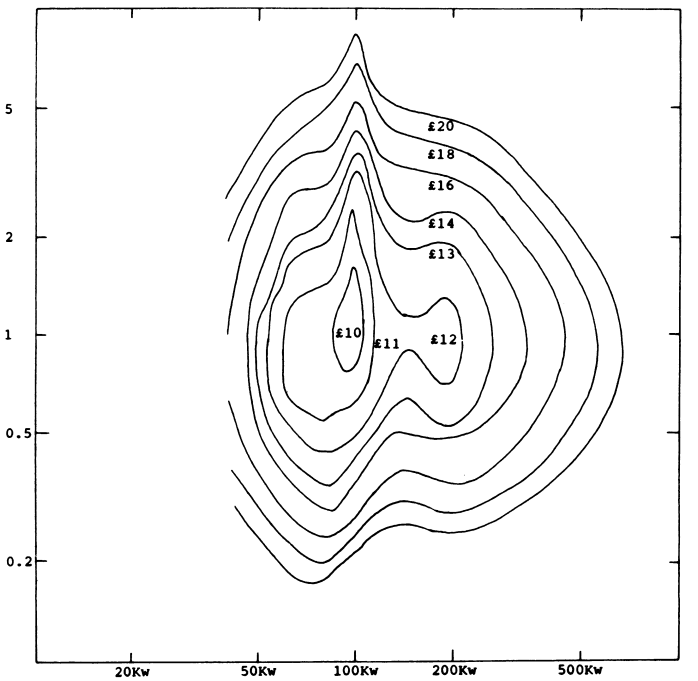

Fig. 8 Estimated cost per job for University workload

5. Results of initial experiments

5.1 *Cost/performance analyses*

Using initial approximations in all the areas discussed in the previous section, a number of performance values have been explored over a range of store sizes and CPU speeds. The graphs shown in Figs. 2 to 8 are to a logarithmic scale in both directions, the store scale (horizontal) varying from 10 to 1000 kilowords, and the CPU speed scale (vertical) from 0.1 to 10 MIPS (million instructions per second).

The curves of constant cost configurations are shown in **Fig. 2**—obviously the more store one purchases, the less cash is available for the processor. The curves of constant performance configurations for 100 jobs of a simple workload (uniformly distributed between 10 and 20K store demand, and 1 and 5 minutes elapsed time on a 1 MIP machine) are shown in **Fig. 3.** The time quoted on each line is the total time to complete the workload (first job loaded to last job ended).

The optimum performance for a given cost of hardware is now the locus of points where a curve of Fig. 2 is tangential to a curve of Fig. 3. This locus is shown in **Fig. 4.** We can also use the curves of Figs. 2 and 3 to produce contours of cost per job figures. These are shown in **Fig. 5**, and show graphically the variation of cost/performance over different configurations.

Figs. 6, 7 and **8** show the same information as Figs. 3, 4 and 5 respectively, but for a workload of statistical distribution approximately that of the University Computing Centre workload at Nottingham. The discontinuities below about 110K of core store arise from the fact that the scheduler is forced to abandon the larger jobs of the workload when the core store falls below this size. Because these are large jobs, to abandon just one of them makes a significant 'apparent' change in performance.

5.2 *Student use*

One of the original intentions of this software was to enable students to obtain a feel for job scheduling and machine configurations relative to performance.

The areas which the model brought home most vividly include:

(*a*) problems of store jams—each job involves a main store overhead, and it is all too easy for jobs to be loaded and swapped out until there is no space left to swap them in again

(*b*) problems of long jobs related to the mean time between system breaks—when the total execution time of a long job becomes of the same order as the mean time between breaks, jobs which do not have restart points never get completed.

6. Conclusion

The system written has achieved its primary objective, that of simulating realistically a varied job mix, using acceptably small amounts of computer time (approximately 1 minute of ICL 1906A mill for 1000 jobs). The system has further shown itself to be capable of extension to any job mix, scheduler and hardware system, provided the appropriate statistics are available.

It is felt that an important development of the model is in the area of user involvement. The model is capable of being extended to include aspects of user satisfaction/dissatisfaction and user responses (for example, repeated resubmission of development jobs a certain time after completion of a test run). The whole area of user relationships with computing centres is generally under-researched at present, yet is a vital consideration in the provision of a service.

7. Acknowledgements

Thanks are due to my colleagues, particularly Dr C. D. Litton, for advice during the development of the above ideas, to the University Computing Centre for workload statistics collected for me, and to my students for testing the resilience of my system.

References

BELADY, L. A. and KUEHEN, C. J. (1967). Dynamic Space-sharing in Computer Systems, *CACM*, Vol. 12, pp. 282-288.

BERNERS-LEE, C. M. (1972). Three Analytic Models of Batch Processing Systems, BCS Conference on Computer Performance, University of Surrey, Sept. 1972, pp. 43-52.

BRANDWAJN, A., BUZEN, J. P., GELENBE, E., and POTIER, D. (1974). A Model of Performance for Virtual Memory Systems, *Proc. ACM SIGMETRICS Symposium*, Sept. 1974.

BUZEN, J. P. (1971). Analysis of System Bottlenecks Using Queuing Network Models, *Proc. ACM SIGOPS Workshop on System Performance Evaluation*, April 1971, pp. 82-103.

CHEN, P. P. (1975). Queuing Network Model of Interactive Computing Systems, *Proc. IEEE*, Vol. 63, No. 6.

CHOW, W. M. (1975). Central Server Model for Multiprogrammed Computer Systems with Different Classes of Jobs, *IBM J. Res and Dev*, Vol. 19, No. 3, p. 314.

COFFMAN, E. G. and MITRANI, I. (1975). Selecting a Scheduling Rule that Meets Pre-Specified Response Time Demands, *Proc. 5th Symp. on Operating Systems Principles*, Austin.

COURTOIS, P. J. (1971). On the near-complete-decomposability of networks of queues and of stochastic models of multiprogramming computer systems, Comp. Sci. Dept. Carnegie-Mellon University, Pittsburgh, Pa., Rep. CMU-CS-72-111, Nov. 1971.

GAVER, D. P. and SHEDLER, G. S. (1971). Multiprogramming System Performance Via Diffusion Approximations, IBM Research Report RJ-938, Yorktown Heights, NY.

KIMBLETON, S. R, (1974). A Fast Approach to Computer System Performance Prediction, Conference on Computer Architecture and Networks, IRIA, France, August 1974.

KOBAYASHI, H. (1974). Application of the Diffusion Approximation to Queuing Networks, I: Equilibrium Queue Distributions, *JACM*, Vol. 21, No. 2, pp. 316-328.
MITRANI, I. and HINE, J. H. (1975). Complete Parameterised Families of Job Scheduling Strategies, Technical Report 81, Computing Laboratory, University of Newcastle upon Tyne, October 1975.
SALTZER, J. H. (1974). A Simple Linear Model of Demand Paging Performance, *CACM*, Vol. 17, pp. 181-185.
SALMAN, O. (1976). Mathematical Modelling of Computer Systems, Ph.D. Thesis, University of Nottingham.

Book reviews

Managing Software Projects, by J. K. Buckle, 1977; 108 pages. (*Macdonald and Janes*, £3·95)

This is a slim volume in the series of 'Computer Monographs' published by Macdonald and Jane's, written by British authors, mainly I suspect with experience in one way or another of ICL machines and methods. Mr Buckle, who uses his experience in ICL for the basis of his 108 pages, does provide some good ideas on writing software to a planned budget for anyone who is totally new to the subject. But for a prospective reader who has experienced the frustrations of actually trying to control large scale program development, he provides little that is new or concrete and tends to make do with vague generalisations in the really tricky areas.

The author defines software as being not only computer manufacturers' compilers and operating systems, but also a user's implementation of a complex data base; or even a suite of programs based on a single set of files where intercommunication is important. But one has the feeling that his heart is really in writing the manufacturer's software and the environment he describes and the methods he proposes seem less suited to a normal DP installation—see the discussion on 'Releases', 'Phasing' and 'Implementation Tools'.

Mr Buckle does not seem to cover the function of the systems analyst and omits his investigations and work on economic justification, in order to concentrate on the job of the project leader of the programming team. This project leader tends to be depicted not only as all powerful in controlling the resources, but to be given the task of producing software without any real need to justify the project or its total cost. In the real world of commercial DP, estimates of programming cost and time have to be made at the initial proposal stage, long before the task is well enough defined to suit the techniques of Mr Buckle's project leader. If the project is given the go-ahead, then it is these initial estimates which tend to form the project leader's budget—whatever his subsequent investigations may indicate. Also the project leader has to compete for programmer resources with all the other projects and suffers from staff leaving and user departments changing their mind. The author says 'find a single person who can represent the user departments, and carry out all negotiations with him', but most 'customers' in firms would not be willing to entrust one person with their requirements, and desperately retain the freedom of individual bargaining. Even if one user by sheer willpower becomes the acknowledged negotiator, the odds are he will be promoted or transferred before the project is completed.

The author describes the typical documentation required to record and control the project, such as the 'Functional requirements specification', 'File descriptions', 'Module maps', 'Testing plans', 'Project log', etc. All these are of course worthwhile and one idea, the 'Slip Chart'—to show the different estimated times of completion at various stages of the project, measure slippage and record action—is a useful addition. But the crucial problem still arises as to what to do when there is slippage? What actions are possible if the estimates of programming time and complexity were wrong? What can be done if there are no spare programmers or less high priority projects, or no extra machine time available? How can you accurately measure progress? Mr Buckle is not very helpful, especially as he seems to rely on everyone acting logically, unemotionally and with great ability.

The difficult process of estimation of programming time and resources also receives little assistance in the book, reliance being placed on basing estimates on the number of lines of code per average programmer per unit of time. This seems to leave large scope for errors, especially in deciding how many lines of code there will be in the first place. Estimating techniques for software still seem to be primitive in most installations, including from this evidence, computer manufacturers.

Mr Buckle likens the development of software to the initial design and prototype construction of a car or a computer. This may be acceptable to 'customers' in computer manufacturers who will realise the problems of prototypes, but it will not be accepted by the average department manager of a business concern, who tends to be sceptical about using a computer system in the first place and needs reassurance that the program will really work rather than be asked to make allowances for the hazards of developing a new product. The computer profession must be able to do better than this or the potential benefits will never be accepted in principle nor obtained in practice.

To sum up, this is an important topic for the computer industry and the book is easy to read and worth perusing. But it cannot be recommended as the solution to the problem, least of all in commercial DP.

R. M. PAINE (London)

The State of the Art in Numerical Analysis, edited by D. A. H. Jacobs, 1977; 978 pages. (*Academic Press*, £20·00)

This book is based upon the proceedings of a conference of the same title organised by the Institute of Mathematics and its Applications in April 1976. The first residential conference of the IMA was held in 1965, also with the same title and the proceedings of that conference were also published in book form (*Numerical Analysis: an Introduction*, edited by J. Walsh, 1966; 212 pages. *Academic Press*). Dr Scriven notes in his foreword to the present volume that 'since the conference the use of computational methods to solve the mathematical problems arising from a great variety of application areas has exploded in keeping with the availability of increased computing power.' Thus, whereas the aim of the earlier volume was to provide 'a general survey of the principal topics of interest without attempting to cover them fully', the present 'volume surveys the theoretical and practical developments across a wide area of numerical analysis over the past decade', and fully justifies its title. The material in the book is presented in seven distinct parts which, with the exception of Part III which is the longest and contains five chapters, contain three chapters. The topics covered in these seven parts are I Linear algebra, II Error analysis, III Optimisation and nonlinear systems, IV Ordinary differential equations and quadrature, V Approximation theory, VI Parabolic and hyperbolic problems, VII Elliptic problems and integral equations; each of the 24 chapters is written by a specialist in the field. A feature of each chapter is the substantial number of references given. The book is handsomely bound and printed on good quality paper, although in the interests of economy the printed page is in the form of typescript photocopy. The price is probably sufficiently high to deter private subscription, but anyone who is interested in recent theoretical and practical developments in numerical analysis should ensure that he has access to a copy.

N. RILEY (Norwich)