Evaluating Computer Systems Simulation Models

Constantine Lazos

Department of Mathematics, University of Thessaloniki and Graduate Industrial School of Thessaloniki, Greece

Abstract

The need to validate and evaluate the running of simulators of computer systems is considered. The paper describes the systematic construction of a simulator to minimise logical and modelling errors and discusses three alternative ways of validating and evaluating the correctness and accuracy of such simulators. The advantages and disadvantages of the method are outlined.

INTRODUCTION

The term simulation and the associated concept of modelling are used in many different contexts. In particular, one can contrast continuous or discrete systems and hardware versus software simulations. Continuous systems are usually deterministic and can be described by differential equations or time series. Discrete systems are often stochastic, involving queues and scheduling strategies and do not lend themselves easily to explicit mathematical formulations. Special purpose languages such as SIMSCRIPT (1) and GPSS (2) have been developed to facilitate the construction of discrete simulation models. More general purpose languages such as FORTRAN (3,4), ALGOL (5) and others (6) have also been used. The value of simulation lies in the fact that simulators are in effect 'artificial laboratories' and as such enable analysts to 'get insight' and understand how systems will behave under a variety of operating conditions, without disturbing the actual system which in many instances would be costly or even dangerous.

Simulation has been successfully applied by analysts to such diverse fields as engineering and economics and it has been proved to be a valuable tool for attacking many evaluation problems. However, simulation suffers from several disadvantages, including the irreproducibility associated with the simulated results (7), potentially a long development time and debugging (8) and validation of the simulator program itself (6,4,9). Because simulation represents a given system (or class of systems) its complexity clearly depends on the complexity of the system(s) itself.

SIMULATION OF COMPUTER SYSTEMS - NEED FOR EVALUATION

Computer systems today are amongst the most sophisticated and complex ever built. In particular, large configurations are very complex environments to simulate. Even the most modest of computer systems is a highly interdependent amalgamation of hardware components and software algorithms. The simulator for a computer system must include three models: the hardware model, the software model and the model of programs (jobs) behaviour. Thus, the analyst needs to combine these three models into one and in doing so accommodate their interaction. This clearly indicates the difficulty in designing and building such simulators.

Present Address: Computer Centre
Academic Division
University of Birmingham
England

CH1437-3/79/0309-0316$00.75

1979 Winter Simulation Conference

There are three distinct phases in developing a simulator for a computer system. Phase one deals with the formulation of the model i.e. what features of the system are going to be included in the model and how important or unimportant are those left out; in other words at what level of detail the model is going to be built. The model, as a representation of a real system, cannot include all the functions, parameters characteristics, etc., of the system. Indeed, models should not be constructed as "icons" of the real system because such models are very costly and consume a lot of operation resources such as human effort, execution time, space etc. (13) Phase two is the realisation of the design of the model via a computer program and phase three is the preparation of suitable data to drive the simulator. Though phase two and three are more time-consuming than phase one, it is phase one that is the most important. A bad design of the basic model could create a lot of problems for the other two phases particularly if a major redesign of the basic model is needed. This may invalidate most, if not all, of the work done for phase two and three. Nevertheless once the basic model has been formulated and phase two is completed, the simulator, like all programs, may produce wrong results. The wrong results may originate from two different sources:

(a) Logical errors.

(b) Modelling errors (i.e. an ill-designed model, omission of details of disproportionate influence of one or more parameters on the results).

The first type of error may show itself up by grossly unexpected results and it is rather easy to spot and correct them. The other type of error is very difficult to detect due to the complexity of the system the model represents. The question hence is, how one can be sure that the simulator is free of modelling errors so the results can be interpreted with confidence and so any alteration of the model's parameters to study the effects on systems performance would produce results that too could be interpreted with confidence.

The analyst may well believe that everything has been carefully designed and built, but the human brain is a very unreliable device and though the results may look logical, in fact they may be inaccurate. Thus the only way to make sure that the simulator is free of both logical and modelling errors, is evaluation.

METHODS OF EVALUATION

The development of a simulation model is only the first step of the analysis process. The next step is to test and validate the model. Only then can analysis and interpretation of results be performed with confidence.

There are three basic ways of testing a model:

(a) by comparison of simulation results with data obtained from the actual system;

(b) by comparison of simulation results with results obtained through analytical methods (degenerate cases);

(c) by substituting fixed values for job characteristics as well as for other parameters, instead of random samples.

We are going to analyse these three models (in reverse order) and report from our own experience in developing a simulator for a real system.

The system in question was based on an ICL-1907 computer, and its associated multi-programmed operating system. The model developed, simulated the whole system, i.e. the off-line input of jobs, the processing of them and the off-line output. The model was structured on the next-event type simulation. Events are created by jobs arriving in the system, jobs being selected for execution (by time slices) and by the start and finish of transfers to/from the peripheral devices (disc, tapes, etc.). Events occurring in much shorter time intervals such as memory accesses were ignored. Thus, the model was a gross model or of the MACRO-simulation type. Also, although the model was simulating a specific system it was designed in such a way as to leave it independent of the 1907 machine configuration, and permitted the inclusion of specific software algorithms, which are at present in any comprehensive multi-programming computer system.

1. Using Fixed Values

When it had been decided at which level the system would be simulated and

what the model should include it was constructed in a modular fashion. (A module being a collection of related system functions). Then each module was carefully 'screened' (as described below) before being interfaced to other modules so that the development of the system could be controlled and structured for testing etc.

During the development of each module a series of fixed values were substituted for both the job characteristics and for system parameters. For example, the number of I/O requests, amount of CPU time the job would absorb, CPU time slices, were set to constant values as were system parameters such as the time to transfer a record from/to disc. This made it easy to check the results against hand calculated values. It was also found to be very helpful in detecting and correcting logical errors by producing a step-by-step trace of the model; again of considerable help in debugging the modules and the complete system to such a degree that when the next stage arrived we were surprised to see how well the system performed.

However, though fixed values were of considerable help in checking the logic of the simulator, we felt they alone cannot solve the problem of validation (i.e. modelling errors).

2. Testing the model against analytical models

Our next step was to test the model against an analytical one. What we needed was to formulate an analytical model of the real system or use such analytical models that have already been developed. Ideally, the analytical model should cover (or represent) the same system as to permit a direct comparison between the two sets of results. Formulating such an analytical model (i.e. to cover the entire computer system) was found to be very difficult, so we decided to use an analytical model that was already built (11) concerning CPU utilization and we also built another model concerning average queue length for the jobs waiting for the CPU (9). Both these analytical models had an unlimited number of I/O channels in contrast to the simulation model which had a limited number. Clearly the results of the two models could not be compared. Hence, we decided to bring the simulator model closer to the analytical representation by suppressing certain features of it. In doing so we have departed from the basic simulation but still we found that certain paths and certain design features of the simulator (mainly queues) could be tested, which was not possible with the fixed values method.

Though the disadvantage of this method is that in most cases the analytical model is a subset of the simulation one, it is still worth it because certain vital paths of the simulation can be tested thus making a considerable contribution to the validation of the simulation program (6,9,15).

3. Simulation Results v Real Data

If the simulation model under consideration represents an existing system, as in our case, the best way of testing the model was to attempt a direct comparison of the simulation results and those measured of the real system.

To test the model against real system data it is necessary to collect the data and then use this data in order to generate virtual jobs to drive the model (see fig. 1). Collecting and analysing the data can be a tedious and time-consuming process. Many computer systems keep accounts of each job and record various events (of each job they process), such as the time the job started, time finished, etc., system resources the program is using, i.e. CPU time, core memory, etc., are also usually recorded. If the information gathered by the system itself is adequate for the model then collection of system data is an easy process. If, on the other hand, the model requires more detailed information to drive it, data acquisition becomes more difficult, if not impossible, and the solution may only be achieved by estimating parameter values, or by suppressing such parameters thereby making the model more approximate. In extreme cases it may become necessary to modify the operating system in order to obtain the necessary data. Clearly the dichotomy between the details incorporated into the design of the model and the costs and effort to operate the simulator, is again evident.

Hence, we first acquired the system log-tapes (tapes on which the system monitors information about jobs, resources they use, etc.) and analysed the data. The analysis showed that the data collected were far from being satisfactory for our model, for example job priority was not recorded and it was not known which of the two line printers was used to output the job. The time the job started execution and the time it was finished (elapsed time) was not recorded. Data such as these were very important both for creating virtual jobs and to have real data to check the simulators results with the actual system. Obviously we could gear the model to meet the specifications and be tested with those data collected already by the system, but a close inspection of this revealed that, either the model had to undergo many

alterations or again as with the analytic case, only certain parts of it would be tested. More data was therefore needed, this simply meant that the system had to be modified. This was terrifying because modifying the operating system was very difficult and dangerous and could easily cause an upheaval to the users. Luckily, the O/S was developed by the local staff of the computer centre which meant that they could easily do the job. Unfortunately, when we approached them (after approval by the department) we found they were not very willing to make the changes, partly because they did not want the simulation to tell them that the system could perform better if certain changes were incorporated. It took us quite some time and a lot of persuasion to convince them to do the job. It took them about 4 man-months time to do the changes and start collecting statistics.

The data collected were not in a suitable format to drive the simulator. Modifications and more important analysis of the data necessary to the model was required on the "raw" data. At this point we had to decide how the data should be analysed to make it suitable for the simulator. We had two options (a) to create individual job profile and use the TDM method (14); and (b) to statistically analyse it and derive appropriate distributions either theoretical or empirical ones.

Of the two alternatives we used the first one, firstly because we felt it was more natural and secondly, much of the collected data were biased, by operator intervention or hardware malfunctions (see below) and no good for statistical analysis.

3.1 Job profiles

The recorded data were scanned and all measurements of the same job were brought together to form the job's individual profile. For example

Time the job commenced was read in;
Time the job was selected for execution;
Job priority or class;
System resources - job requested;
System resources - job actually used;
Time the execution of the job finished;
Device on which it was printed/ punched, together with the time started and time finished, etc.

Thus a vector of values was created which included job characteristics, job behaviour and system measured performance. This vector is called the individual job profile. Because the system was multi-programmed, the job profiles include the effect of multiprogramming on each job. This was found to be of primary importance (4) for comparisons with the simulation results.

The traced data for every job and for the system were accepted as a faithful reproduction of the actual workload and system performance, provided there was no operator intervention or any hardware malfunction. Consequently, job profiles were used directly to drive the simulator model. Thus instead of sampling statistical distributions to generate job characteristics the job profile was read and its actual characteristics were used. In this way exactly the same load as that of the real system would go through the simulator.

This method permitted direct comparisons of the simulator's and the system's performance on individual jobs. Overall measurement and comparisons were easily obtained at the end of the simulation runs.

Though everything seemed to be alright, the first shock came with the simulation of the first set of jobs. The result of the simulator and those measured of the actual system compare very badly indeed. We turned to the program and carefully checked it. Two modelling errors were found, one in the switching phase of the jobs between compilation and execution. Both were corrected and the run was repeated. Unfortunately, the new results again were not satisfactory. Then we turned to the data. A close examination revealed operators intervention and in some cases system's malfunctions. For example, the system was forced to select jobs for execution ahead of others instead of leaving the system to do it according to its built in scheduler or certain jobs have been accelerated i.e. giving top running priority thus interrupting all other jobs and getting the CPU at any cost. Also some problems with the interchangeable discs were noted as well as with the card reader and line printer, namely card wreckage or printers run out of paper or the need to reprint jobs. We therefore decided to drive the simulator with data collected from "supervised" session, i.e. by permission we stayed in the computer room for a short period of

time (½ - 1½ hours) and asked the operators not to intervene with the system or to make sure that the line printer would not run out of paper in the middle of a job. Thus some data free of any operator intervention or system malfunction were collected and the simulator was run again. This time the results were compared very good. For long sessions it was not possible to supervise them, though the operators were asked not to intervene with the system, there was not guarantee the card reader or the line printers would not run into trouble. Thus, for long sessions only the data of the execution phase were compared. The results compare good again but still we had to search a lot and sometimes to check manually the data to get some free of interventions and malfunctions. In many cases, for example, a long delay caused by the operator mounting tapes or changing disc packs was observed, thus altering the sequence of jobs running.

3.2 Statistical Analysis

In this case, the real data could be analysed and distributions with mean values be calculated (6,10). The purpose of these distributions is to obtain information characterising:

(i) the job stream to be input to the simulation model;

(ii) the treatment given to the jobs by the real system;

(iii) the system performance.

Typical statistics for (i) could be, the inter-arrival time of jobs in the computer with their associated distribution and mean value, and classification of jobs into several classes according to mean values of theoretical or empirical distributions calculated for various job characteristics. For instance, Boot (10) has classified jobs into four classes according to their compute time; then for each class the mean compute-time followed by the distributions within each sub-classification. Such data could be input to the model where a special routine, the "jobs generator routine" utilises these statistics to determine details of the job stream, i.e. derives job characteristics for each job. Similarly during the simulated execution of the jobs, their statistical distributions and their measured mean could be used by the model to predict job behaviour, hardware response time, etc.

Finally, the results of the statistical gathering routine (simulation) could then be compared with the statistics gathered about the real system, i.e. (ii) and (iii) above; in particular, by comparison of mean values (6,10).

ADVANTAGES AND DISADVANTAGES

In order to derive statistical distributions for job characteristics, analysis of the collected data involves:

(a) the collection or trace of that data concerned with individual jobs; and

(b) the formation of distributions by further classification.

This implies that apart from the programming effort needed, one has to have advanced knowledge of statistics and sampling theory.

In contrast, the other method of analysis, the TDM method, requires less programming effort to prepare the job profiles for a run and requires virtually no knowledge of statistics or sampling theory.

On the other hand since the model has been tested against real data and it is found to perform satisfactory, then one may wish to run the simulator again by altering the workload of the system to see, for example, what effect on system performance would be an increase of say the small jobs. Clearly this can be done if statistical distributions for generating the workload are used while with the job profile method of TDM it is more difficult to achieve. In other words, the statistical distributions included can give overall images of the job characteristics and they are more flexible in generating alternative workloads. The job profiles are static data and do not provide the flexibility required.

Both methods have a common disadvantage that data collected by the system should be free of any hardware malfunctions or operator intervention. Again with the TDM analysis of data, such cases can be easily detected, while with the statistics it is more difficult. In the latter such cases can be absorbed and "disappear" within the overall measurements, i.e. mean values. While in the individual profiles such cases can be seen quite clearly.

Again with the TDM one can do well with few unbiased data sets, i.e. free of any

interventions or malfunctions. While to derive statistical distributions, a large number of unbiased data sets is needed which may be quite difficult to obtain. Operators interventions can of course be overcome if the model incorporates all the possible interventions the operator is allowed to make and the recording of data trace mechanism is capable of recording all such interventions.

Future Work

All simulation studies of gross models constructed so far have attempted to compare the system's simulated performance with the actually observed (recorded), using either statistical distributions or job profiles. In both cases, simulation is not easy but results have been generally satisfactory in comparison with actual data.

It would therefore be quite interesting to use both methods on the same simulation in order to examine and investigate the relevant merits of the methods in computer systems simulation.

GENERAL REMARKS

The simulation program took us about 12 man-months time to develop and to validate. Validation was the most difficult and time consuming process. The program was entirely written in FORTRAN. This is not to say that simulation languages are not good. On the contrary they have been designed to make life easier for the builders of such simulation programs with the benefit of developing the program quicker, thus saving both time and capital. Still we did it in FORTRAN for the following two reasons:

(a) The computer system used did have a simulation language but no-one in the centre had ever used the language. This meant that we would have had to go alone all the way through, i.e. if we run into difficulties with the language and the Compiler no-one could be of any use to us.

(b) The language was tied to a particular type of computer (ICL) thus making its compatibility very limited, while FORTRAN is a more international language. This last point was to be proved later a very important decision we made. When we left the University and were employed in another place the new machine was a UNIVAC one. It took us a couple of days only to adapt the simulation program to the new machine and make it work. Had we written it in the simulation language it would have been impossible to adap it and put it on to the new machine.

This again is not to say that simulation should be written in FORTRAN. Our view on this is that, if the person who develops the simulator as part of his own research is to move from that place he should think twice about the compatibility program before he chooses the language to use.

The auxiliary program to analyse the data and form the profiles was naturally written in FORTRAN as well. The program was composed of more than 700 statements and it was developed within the 12 months period mentioned above. The simulator itself had about 1800 FORTRAN statements and the ratio of its running time to that of the actual system was about 1:8.

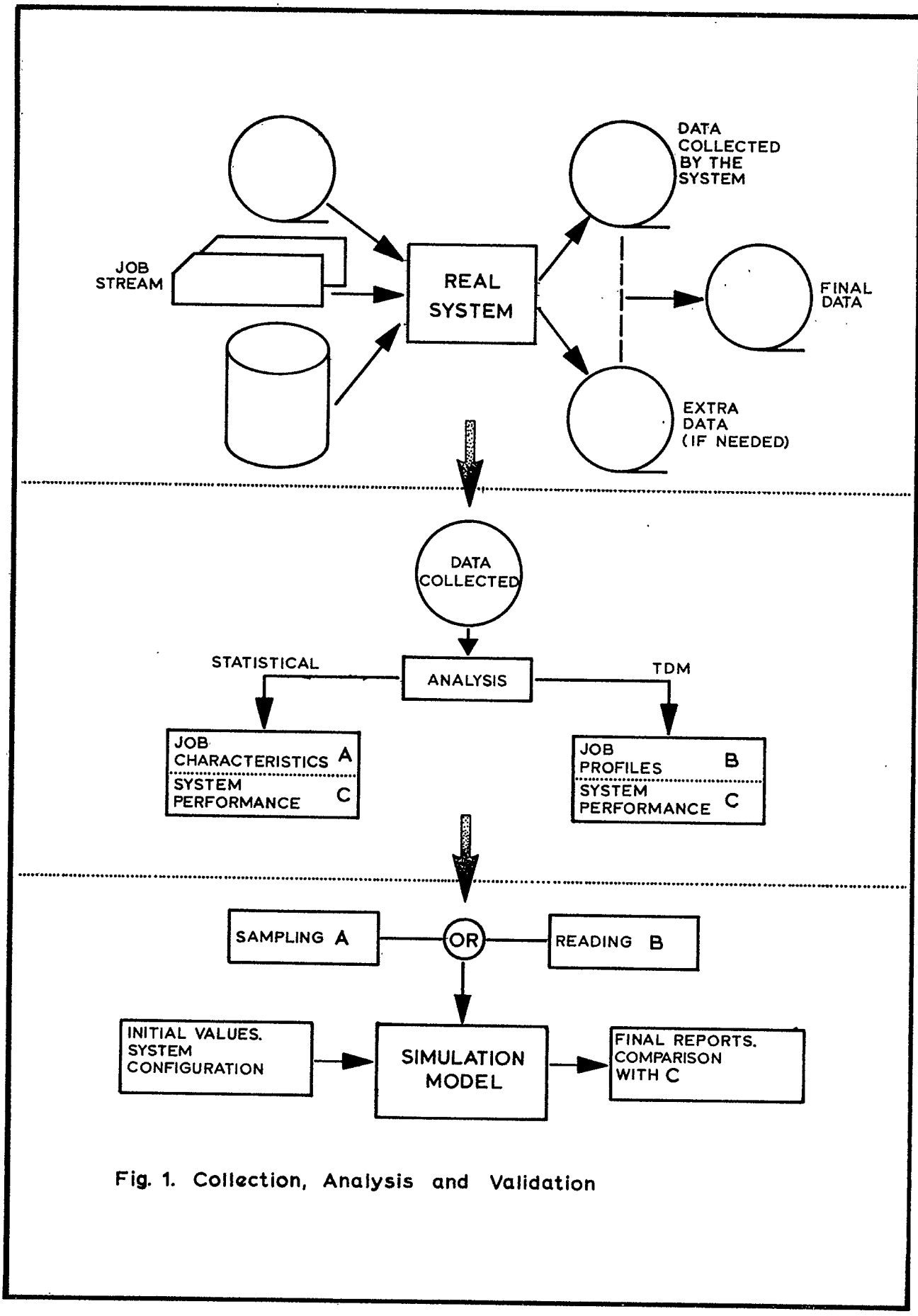

Fig. 1. Collection, Analysis and Validation

BIBLIOGRAPHY

1. Markowitz, H.M. et a., "SIMSCRIPT - A simulation Language", Prentice Hall Inc., Englewood Cliffs, N.Y., 1963.

2. Scheibers, T.J., "Simulation using GPSS", John Wiley & Sons, 1974.

3. Nielsen, N.R., "The Simulation of Time-Sharing Systems", Comm. ACM, Vol. 10, No. 7, 1967.

4. Lazos, C., "Workload of a Linked Computer System - A Simulated Study", Ph.D. Thesis, University of Southampton, 1974.

5. Dahl, Ole-Johan, "SIMULA - An ALGOL-Based Simulation Language", Comm. ACM, IX, No. 9, September 1966.

6. Scherr, A.L., "An Analysis of Time-shared Computer Systems", Res. Monograph No. 36, MIT Press, Cambridge, Mass., 1967.

7. Clark, S.R. et al., "A Note on the Reproducibility of Discrete-Event Simulation Studies", INFOR, Vol. 10, No. 2, June 1972.

8. Nielsen, N.R., "The Simulation of Time-Sharing Systems", Comm. ACM, Vol. 10, No. 7, July 1967.

9. Lazos, C., "A Comparison of Simulation Results and a Mathematical Model of a Multiprogramming System", Information Processing Letters, Vol. 3, No. 5, May 1975.

10. Boot, W.P. et al., "Simulation of a Paging Computer System", The Computer Journal, Vol. 15, No. 1, 1972.

11. Gaver, D.P. Jr., "Probability models for multiprogramming computer systems", Journal ACM, No. 14, No. 3, 1967.

12. Sherman, J. et al., "Trace driven modelling and analysis of CPU scheduling in a multiprogramming system", Comm. ACM, Vol. 15, No. 12, December 1972.

13. Kimbleton, R.S., "The role of Computer System models in performance evaluation", Comm. ACM, Vol. 15, No. 7, July 1972.

14. Cheng, P.S., "Trace-driven Modelling", IBM Systems Journal, No. 4, 1969.

15. Rafii, A., "Study of the Performance of RPS", Performance Evaluation Review, ACSIGMETRICS, Vol. 5, No. 4, 1976.