1988

A Note on Two Simulation Benchmarks

Vernon J. Rego
Purdue University, rego@cs.purdue.edu

Report Number:
88-741

A NOTE ON TWO SIMULATION BENCHMARKS

Vernon Rego

CSD-TR-741
February 1988

Abstract: This note presents the results of a comparative study done on the popular simulation languages SLAM and GPSS. The models that we tested were executed on a CDC Cyber 750, a multiprogramming system with 377000_8 words of main memory and no virtual memory. A single server queueing model and a multiqueuing model for simulating a token ring local area network were used for the comparison. In addition to SLAM and GPSS, we also develop FORTRAN versions of the models to obtain an idea of how a low level model performs in comparison to a model constructed in a well developed simulation language. The note includes a discussion and some graphical results that may be useful to a modelling practitioner in planning a simulation study.

1. Introduction

This note reports the results of a comparison study done on two popular simulation languages, namely GPSS V and SLAM II. The version of the former that we use is actually a CDC version of the IBM release of GPSS V, developed by the Vogelback Computing Center at Northwestern University, and called GPSS V/6000. SLAM II is a descendant of the simulation languages Q-GERT and GASP IV and was created by Pritsker & Associates, West Lafayette, Indiana.

In the following sections we will first briefly describe the two languages and then present the results of our study. The intention is to observe and try to interpret their behaviour in a fashion that may help users plan well, prior to a (possibly expensive) venture involving simulation. All experiments were performed on a CDC Cyber 750 multiprogramming system with a maximum of 377000_8 (counting in octal arithmetic) 60-bit words of main memory.

GPSS V/6000

GPSS (General Purpose Simulation System) is a discrete event simulation language that is based on the process interaction philosophy. It was developed by Gordon (see [1], [2]) in the early sixties and was first available on the IBM 704, 709 and 7090 computers. GPSS is an easily learned and commonly used simulation language. Some plausible reasons for this are its block structured approach and the close similarity between its blocks and the activities one usually encounters in a system that is to be simulated. It was designed with the practitioner in mind, and its compact form allows fairly complex models to be developed with only a small subset of the language (i.e., only a few block types).

GPSS V/6000 allows the use of eight uniform random number generators and one trace driven generator. The generation of random variates from arbitrary distributions requires corresponding user supplied functions (pairs of points corresponding to the desired distribution)

and a GPSS interpolation procedure. A nice feature is the SAVE/READ utility that enables models in action to be saved along with current statistics at some point in time and later restarted from that point. This is extremely useful as a "checkpoint/restart" procedure for time consuming computer runs where, for example, the stability of the model might be an undecided issue.

GPSS V/6000 is written in Compass (CDC assembly language), presumably for enhanced speed, and operates in an interpretive fashion. A GPSS program compilation involves the translation of identifiers into numerics, producing appropriate diagnostics as required. In our environment, a GPSS model begins execution with a field length of 3000_8 memory words and is allocated additional memory dynamically when required. There are three standard memory allocation options for additional memory.

SLAM II

SLAM II (Simulation Language for Alternative Modeling) is a very flexible language [3] that allows simulation models to operate under three different perspectives: process interaction, event scheduling and/or activity scanning. It provides an easy to use network modeling capability, discrete and continuous modeling, and combinations of these.

The network models of SLAM II are similar to GPSS models in appearance (flowchart or language). They are generally easy to develop. The discrete event modeling feature requires user written FORTRAN subroutines that operate under SLAM II processing philosophy to describe and schedule events. The processor undertakes the task of causing the events to occur when scheduled and maintains statistics. The subroutines are interfaced with the processor not unlike the FORTRAN/Compass HELP blocks of GPSS V/6000.

SLAM II provides a variety of random number functions including antithetic variates and a trace driven generator. One may SAVE and LOAD decoded SLAM networks. However, SLAM does not provide a feature for saving and restarting models once execution begins. SLAM II

operates on network models in an interpretive fashion, but models *other than pure network models* require that the user supplied FORTRAN routines be interfaced with the SLAM II code prior to the execution of the model.

SLAM II is written in FORTRAN and operates with a fixed field length at all times. This means that users must estimate their model memory requirements prior to the execution of the model. In the event that a model may need more memory than that provided in the version currently available, it is possible to replace the MAIN program in SLAM II with a user written program in order to redefine the dimensions of file storage areas.

2. A Comparison

The simplest way to make a comparative statement concerning these two languages is to examine their performance on a common task. We choose to view performance in a broad sense, with features including ease of use, speed and cost. Since each language requires that such a task be defined in a different manner, the problem of coding equivalent definitions of the same task is not, in general, an easy one. In the following discussion we present a description of two models and graphical representations of the behaviour of these models coded in GPSS V/6000 and SLAM II on a CDC Cyber 750.

Model I

The first model is a simple model of a single-server queueing station. The customer arrival process is Poisson, and customer service times are independent and negative exponentially distributed random variables (this is called the M/M/1 queue in Kendall's notation) [4]. The first moments of the interarrival and service time distributions are chosen so that the traffic intensity of the system (ρ) works out to be 0.99, thus ensuring a stable but congested system.

Both queuing theory and common sense tell us that the time t taken by such a queueing system to attain steady state depends, in a complicated way, on $d = 1 - \rho$, for $0 < \rho < 1$. As d

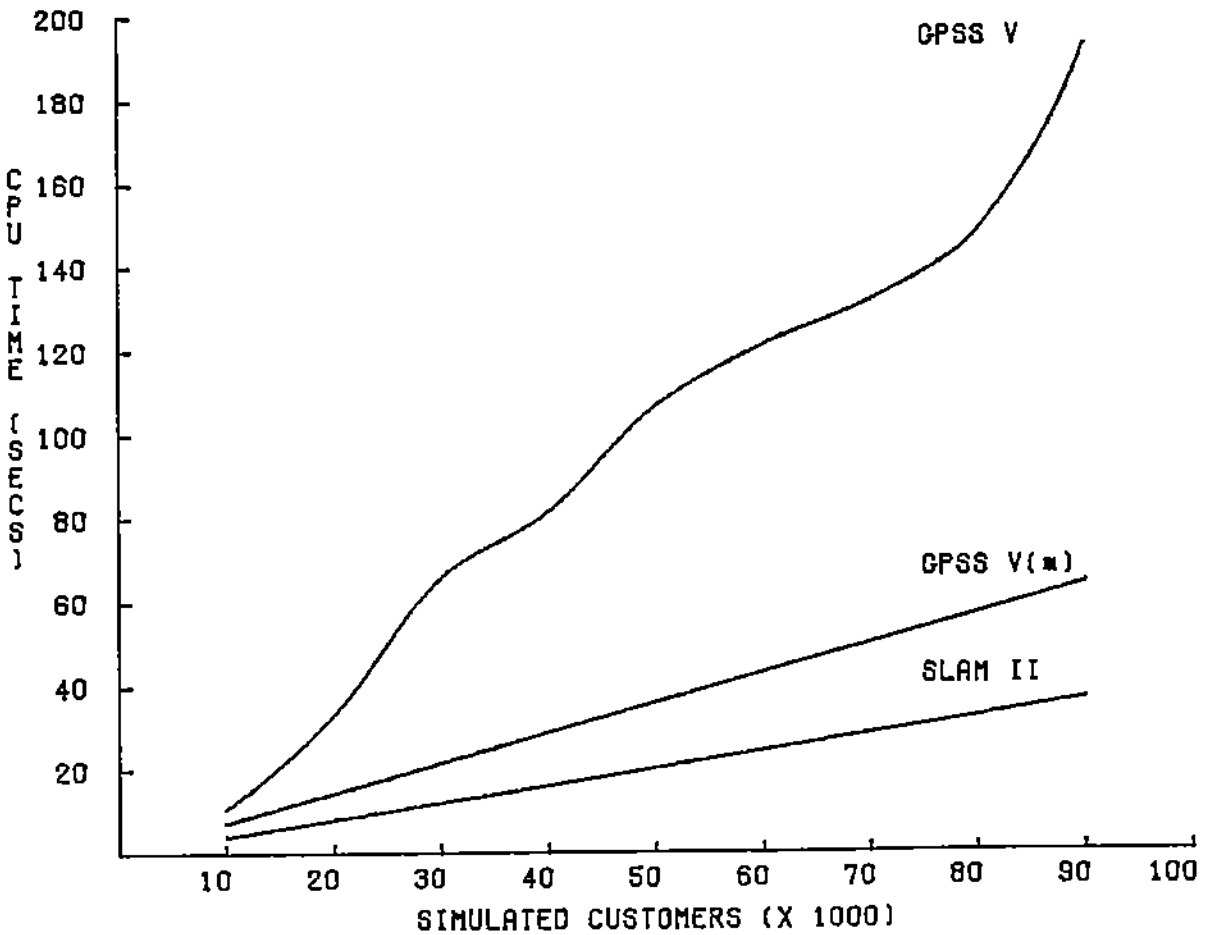

Fig 1a. Execution time for Model I.

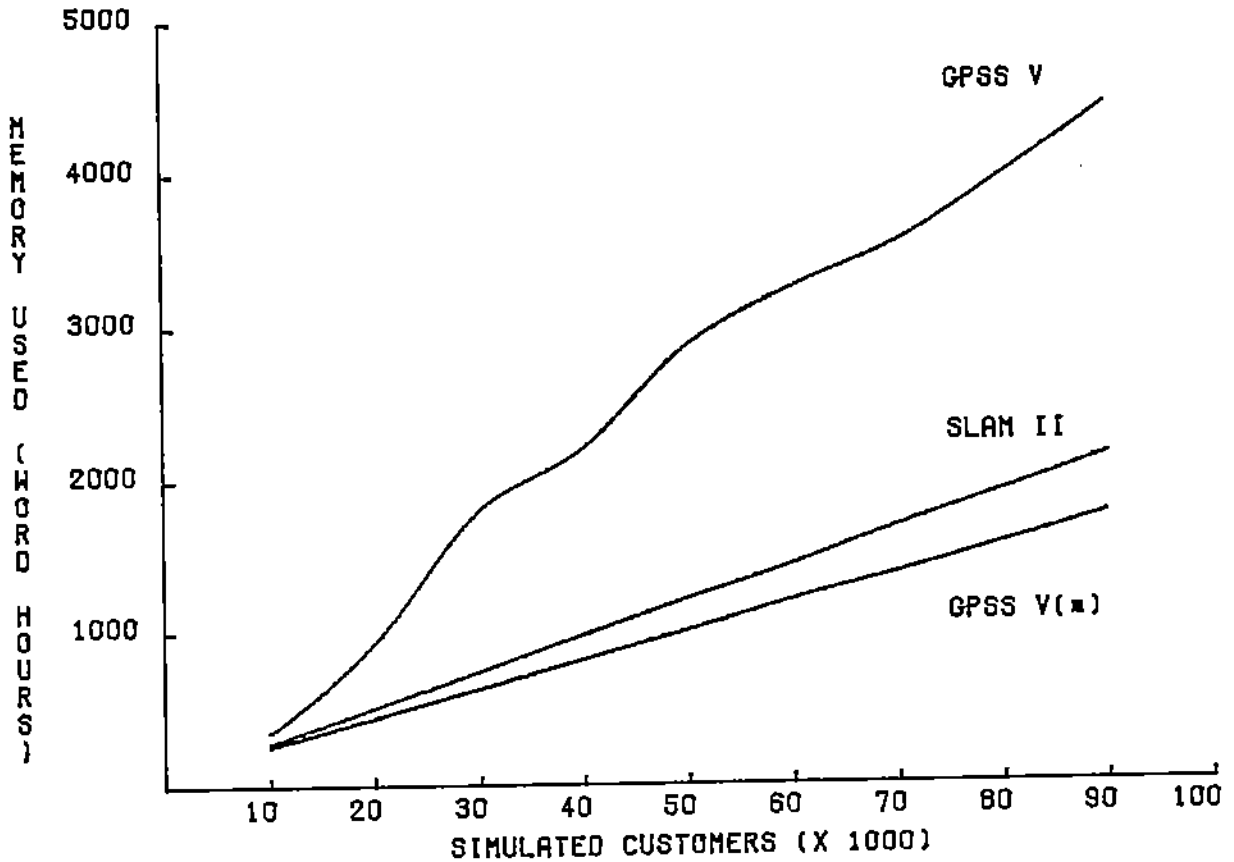

Fig 1b. Memory usage for Model I.

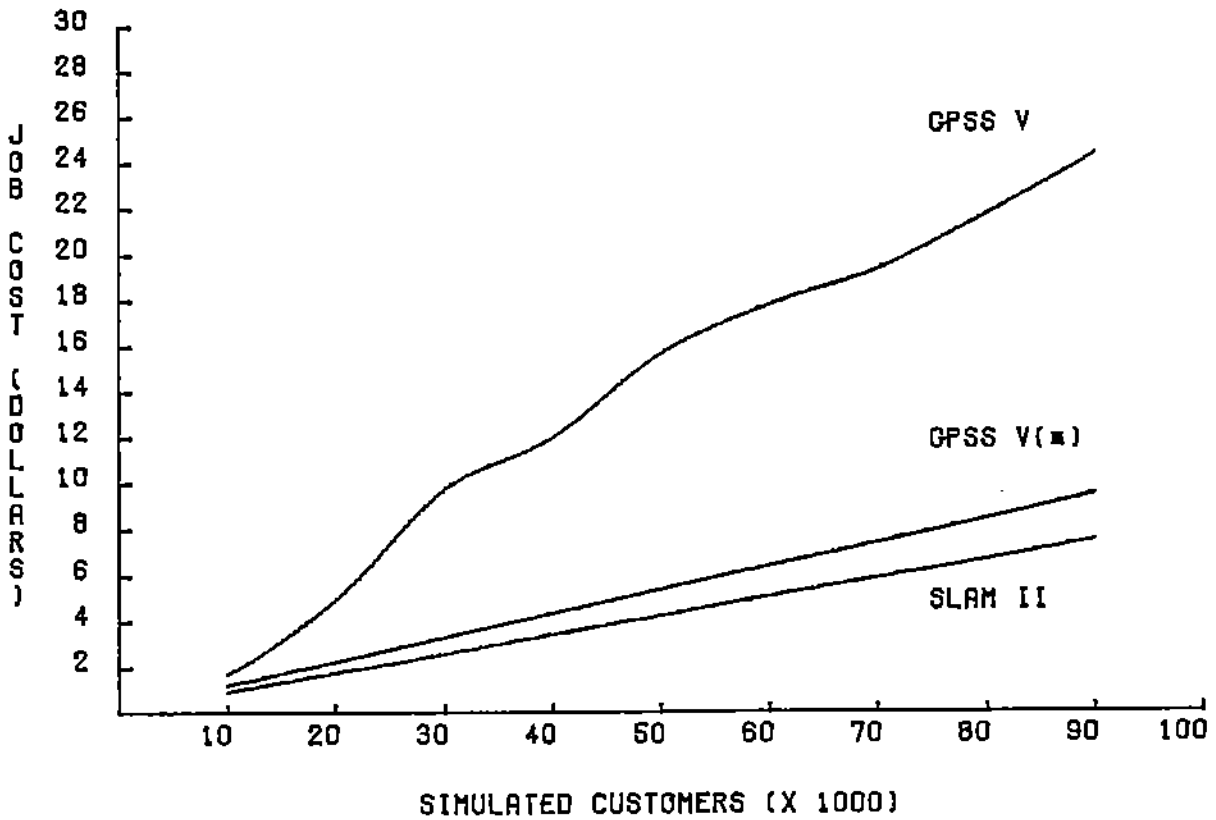

Fig 1c. Job cost for Model I.

decreases, it can be expected that t will correspondingly increase due to rapid increase of congestion in the system. Since the machine dependent simulation models are dependent on CPU attention and workspace (CPU time to replicate observations and workspace to keep track of customers, service, and statistics), a typical simulation run for such a system can be expected to take a time roughly inversely proportional to d in order to attain steady state.

In Figures 1a, 1b and 1c are displayed characteristic results of GPSS V/6000 and SLAM II on the M/M/1 model in terms of CPU time, the amount of main memory used in word hours, and the cost of the job at low priority rates (i.e., batch rates), respectively. Each curve is the result of a set of simulation runs done for a number of customers ranging from 10000 to 90000, incrementing in units of 10000. Also, each curve is obtained through a spline interpolation of the nine points. Figure 1a would lead one to conclude that GPSS V/6000 generally takes longer than SLAM II to perform the same task.

The basic dynamic unit of a model is called a *transaction* in GPSS, and an *entity* in SLAM. A model has a high level of congestion when the number of transactions (entities) in the system is high, and the manner that GPSS (SLAM) scans the current and future events chains (lists) to decide which transactions (entities) should be moved where and at what point in time is largely responsible for the large processing time involved. GPSS makes available a feature that enables users to achieve some reduction in execution time via two blocks. The LINK and UNLINK blocks of GPSS remove transactions from processing (deactivating them temporarily) and bring them into the model when required, thus giving a tremendous potential for reduction in execution time.

The curve labelled as GPSS V(*) in Figure 1a is the result of a model that makes use of LINK/UNLINK blocks. The nonlinearity of the GPSS V curve is a function of the congestion in the system, demonstrating that processing time increases in a nonlinear fashion as the number of transactions in the system increases. SLAM II behaves linearly due to efficient file entry and

removal procedures. Apparently, SLAM does not provide anything akin to the LINK and UNLINK blocks of GPSS, but instead implements an efficient binary search algorithm that effects considerable reduction in execution time by maintaining pointers to subsets of files which are used for file entry or deletion. This procedure comes in handy for large files that are ranked on attribute value. The search algorithm must be invoked by data input changes to the model to get around the standard file processing procedure. The GPSS V(*) curve, like the SLAM II curve, is linear due to the more stable processing of transactions on the event chains, and demonstrates that some level of sophistication in the use of the language is required if efficient models are to be built.

It is interesting to note that the story is quite the reverse, at least in this situation, when we consider the amount of memory used by the job. In Figure 1b can be seen the disparity between the memory requirements for both languages. The unit of measurement used is a word-hour, which represents the usage of one memory word for a period of one hour. Thus program code that utilizes a field length of 1000_{10} memory words for 30 cpu seconds equivalently utilizes $1000 \times 30/3600 = 8.33$ memory word hours. During the course of a program's execution the amount of main memory used by the program is bound to change, being dependent on its temporal requirements. The word hour usage of a job is the amount of memory used by the job, integrated over the time taken by the job to complete.

While SLAM II works with fixed storage and is relatively unaffected by machine architecture (i.e., virtual memory vs. no virtual memory), the IBM GPSS V version specially recommends use of a LOAD statement in a GPSS model so as to make repeatedly used routines reside in core for the duration of the model's execution to cut down on paging costs. This feature, of course, does not make sense on the Cyber 750 since it operates without virtual memory, swapping processes back and forth between main memory and disk. A typical upper bound of 377000_8 usable memory words implies that it is *highly likely* that certain models, whether built in

GPSS V/6000 or SLAM II, may require so much memory during the course of their execution that the simulation will never run to completion. This is very often a problem with large models. For all practical purposes, such models are useless without drastic changes in their design.

The costs of the different jobs at low priority rates are shown in Figure 1c. Cost is calculated as a function that is dependent on the amounts of CPU time, peripheral processor time (PP time), and word hours used by a job. If T_{CPU}, T_{PP} and T_{WH} are used to denote these three times, respectively, the cost C of a job in dollars computed as

$$C = 700. \times T_{CPU} + 9. \times T_{PP} + 0.0037\, T_{CM} \qquad (1)$$

where the constants reflect the unit costs (at current estimates) for the different types of resources.

Model II

The second model is a slightly more complicated model of a multiqueueing system, built to simulate the behaviour of a token ring network [5]. The system is composed of $N = 10$ service stations numbered 1 through N. Initially the system is started with the server arbitrarily positioned at one of the N stations. The server cycles around the system, serving a customer at station j, $j = 1, \ldots, N$, if a customer awaits service there. If the server finds one or more customers awaiting service when he arrives at station j, he spends a random time X_j serving the customer at the head of the queue, and then moves on to station $j \ mod \ N + 1$, taking a random time W_{j+1} to walk over from station j to its successor. The server spends no time at a queue if it is found empty. All random variables used are assumed to be exponentially distributed.

Customer arrivals at the various stations are independent, and within a station these arrivals follow a Poisson process. Each queue is assumed to have unlimited waiting room. Because service is given to at most one waiting customer at each queue per server visit, the service discipline is called the one-at-a-time service discipline. The parameters for walk, service and interarrival times for the different stations are set unequal, so as to yield an *asymmetric* system.

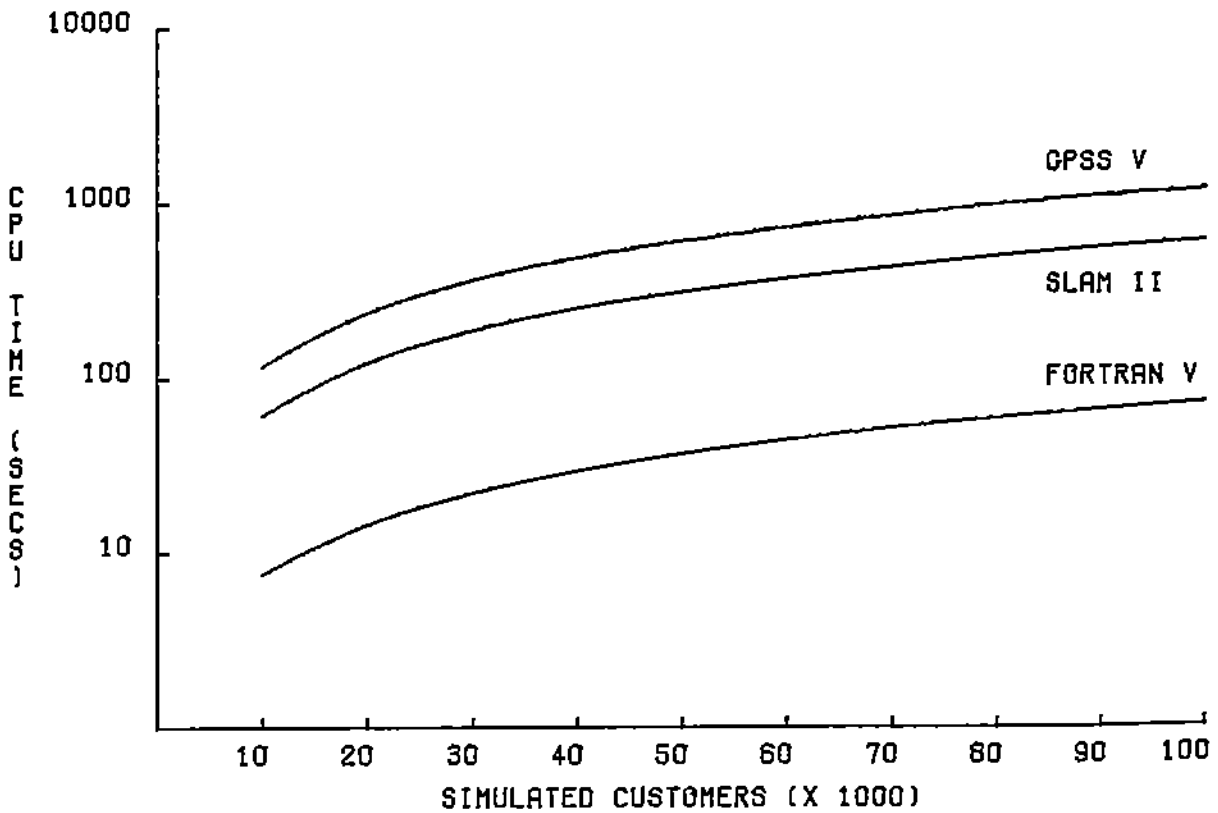

Fig 2a. Execution time for Model II.

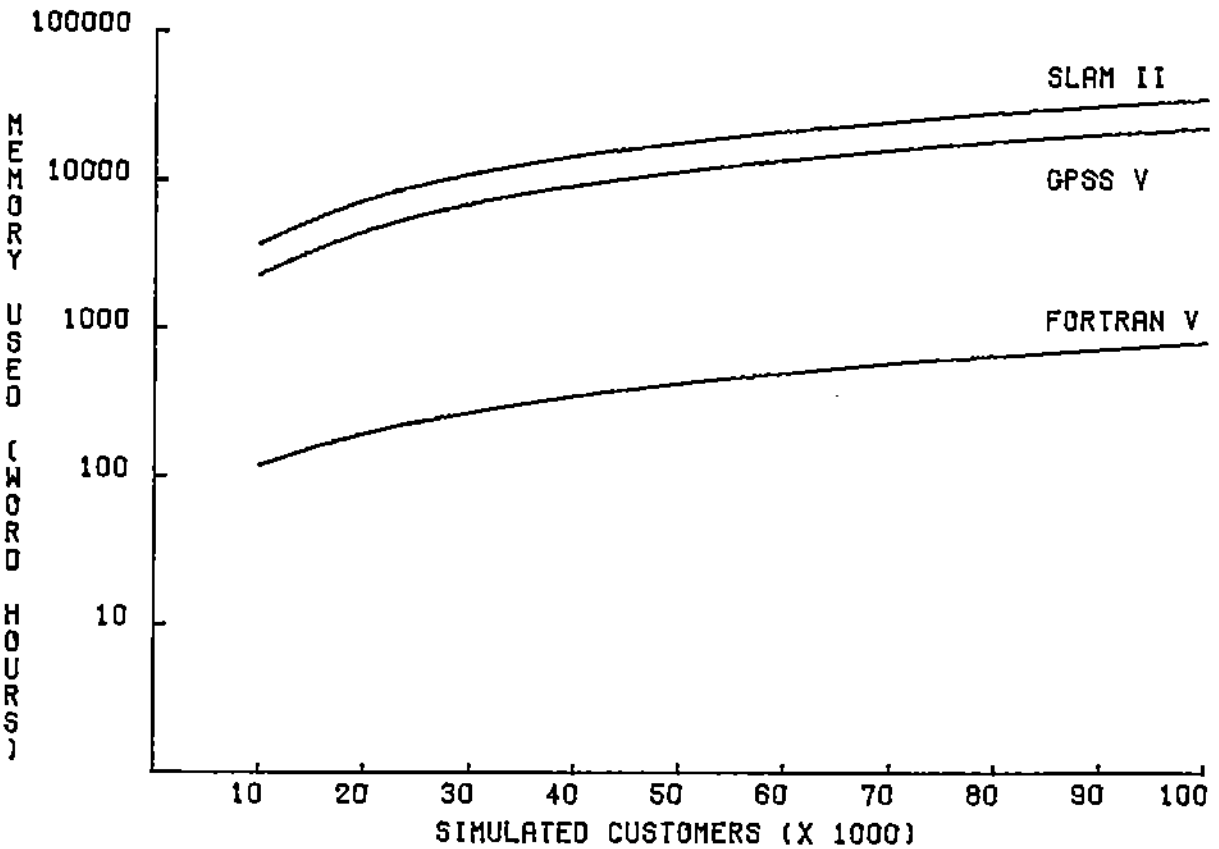

Fig 2b. Memory usage for Model II.

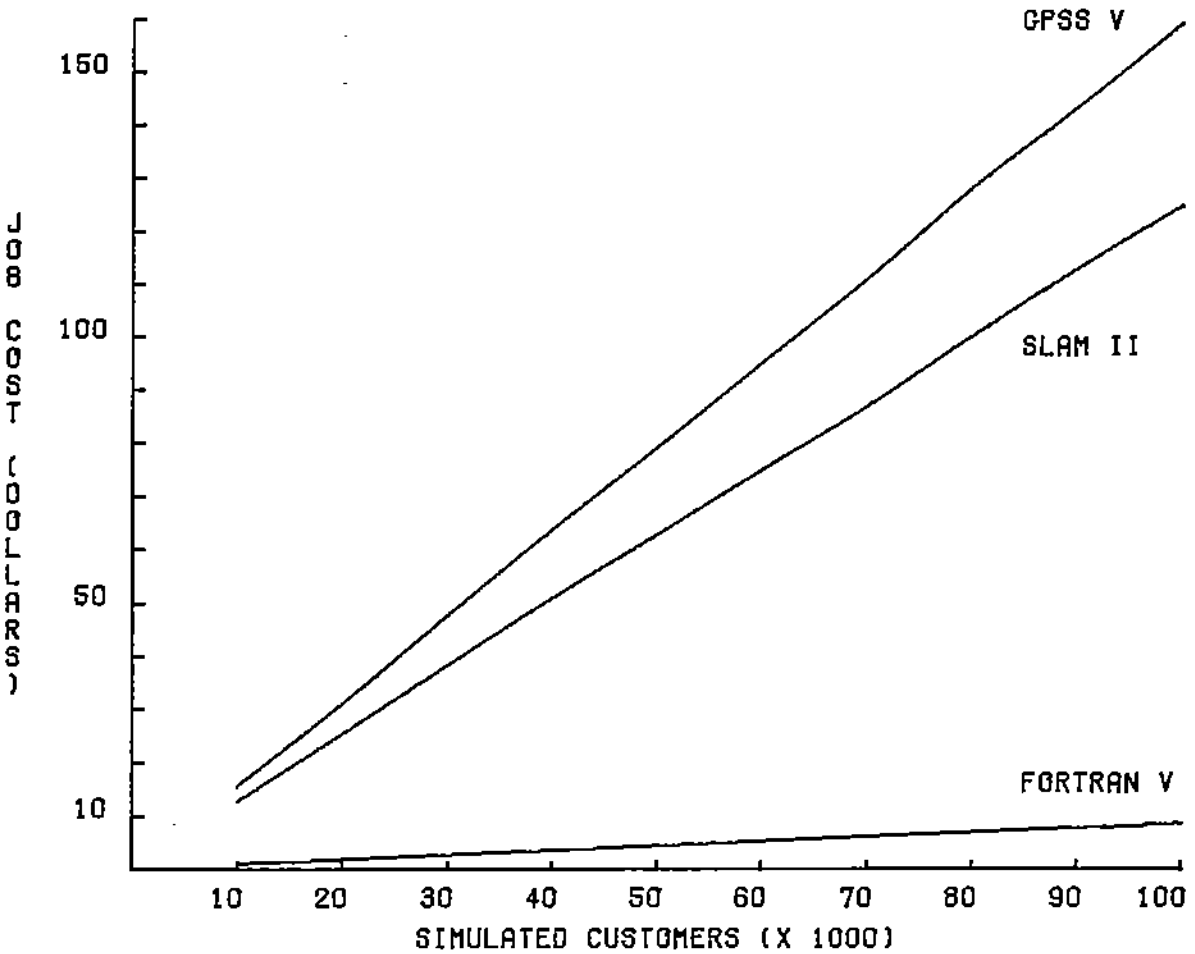

Fig 2c. Job cost for Model II.

For this model, the interarrival and service time means are chosen so that all queues are stable (using the stability condition heuristically obtained in [6], formally proved in [7]) and the value of ρ averaged over all stations is approximately 0.60. It must be noted that no exact queueing solution exists for this model, which makes simulation a necessity. While a traffic intensity of $\rho = 0.60$ can be taken to mean literally no congestion of transactions in a GPSS model, or entities in a SLAM model *for a single server queue*, this *need not be the case* in our multiqueue model. Certain queues may be stable while others are unstable or highly active. We avoid these situations by making all queues stable, with roughly the same traffic intensities, yielding an average value of $\rho = 0.60$.

In Figures 2a, 2b and 2c are displayed the results of this model in GPSS V/6000 and SLAM II, corresponding to the same measures as made with Model I. Further, to demonstrate the costs associated with making such languages easy for non-specialists, a specific FORTRAN V (CDC version) model was also developed, with its results included with the graphs in Figures 2a, 2b and 2c. Such a comparison may not really be fair to simulation languages, especially in view of the fact that coding a FORTRAN (or equivalently, Pascal or C) version of the model can be relatively time consuming, especially for more complex models. Nevertheless, such a comparison makes available all information prior to a decision process involving simulation. GPSS and SLAM required about the same effort for coding, while the FORTRAN model required roughly twice the effort.

Each of the curves in Figures 2a through 2c is a spline interpolation of nine points, increasing in steps of 10000 customers. The customer count is done at a reference queue, and not over the system. Figure 2a once again reveals the nature of GPSS V/6000 in its tendency to require greater CPU attention than SLAM II (with the Y-axis incremented logarithmically in units of CPU seconds). In contrast, the FORTRAN V model takes considerably less time than SLAM II to execute. Due to the fact that congestion in the system is negligible, the use of LINK and

UNLINK blocks in GPSS will be of little avail. It is often the case that in such systems, where a server is required to switch from queue to queue, processing overhead in the model can be very high *in spite of low congestion*. In this case the model executes empty server cycles, with the server scanning queues and waiting for queues to generate customers.

Figure 2b shows the disparity in memory requirements between the two simulation languages and the FORTRAN V model, again expressed logarithmically on the Y-axis in units of memory word hours. It may at first seem a little surprising that the SLAM II memory requirements exceed those of GPSS V/6000, but this can be attributed to the fact that SLAM works with a fixed field length that is larger than the average field length of GPSS.

The FORTRAN V model allows for the most control in memory usage through the use of an efficient technique that would be quite difficult for a package like GPSS or SLAM to duplicate. Simply put, the FORTRAN code utilizes a queue of unit capacity for each station. For each queue only the head-of-line customer's arrival time and corresponding random number seed (i.e., seed that generated this customer's arrival time) need be stored. In order to determine the number of customers queued at station j when the server arrives at queue j, the FORTRAN model calls the random number generator repeatedly until a customer is generated with arrival time greater than the current simulated value of the clock. Note that these times are only generated, but not stored. At this stage the number of customers in queue j is precisely the number of times the random number generator was called minus one. This method of storing seed values and generating quantities only when required is extremely useful, in particular on a fast machine with very limited (and expensive) memory, as is the Cyber 750.

Before a model is executed in SLAM or GPSS, a user must be able to estimate the number of dynamic units that will exist in the model at any given instant. With an analytic or approximate technique to help, a SLAM user may do this in the MAIN program, and GPSS user may specify one of three memory allocation options. In either case this procedure must be followed

for large models, and it is usually here that users will get a feel for whether their models will be treated well by GPSS or SLAM. A program in FORTRAN (or Pascal or C) can avoid this problem altogether at the expense of some planning and coding effort.

In Figure 2c can be seen the total costs associated with the model in the three languages. As expected, a GPSS run costs more than a corresponding run in SLAM. But, as mentioned before, cost is not the only factor. Teaching experience indicates that GPSS is easier for a learner to grasp than SLAM, though the latter may eventually have more to offer. Examples of the latter point are SLAM's continuous and combined modelling capability. GPSS does not require that users know FORTRAN except for fairly sophisticated models that need to modify standard processing by using HELP blocks. On the other hand, SLAM models different from the network models insist that users have a working knowledge of FORTRAN in order that user event routines be developed. Models written only in FORTRAN (or Pascal or C) are usually the most efficient since they can be tailored to suit the user's specific needs. But such models require a great deal of effort and a good knowledge of a programming language. Further, since the user must devise his own filing and accounting system, extreme care must be taken in order that proper statistical results are obtained from the working model.

REFERENCES

[1] G. Gordon, "A General Purpose Systems Simulator," *IBM Systems Journal* , Vol. 1, No. 1, 1962.

[2] *General Purpose Systems Simulator: Program Library*, Ref. 7090-CS-05X, International Business Machines Corporation.

[3] A. Pritsker, *Introduction to Simulation and SLAM II*, Halstead Press, John Wiley, 1986.

[4] A. Allen, *Probability , Statistics, and Queueing Theory With Computer Science Applications*, Academic Press, 1978.

[5] V. Rego and L.M. Ni, "Analytic Models of Cyclic Service Systems and their Application to Token Passing Local Networks," to appear in *IEEE Transactions on Computers*, 1988.

[6] P. Kuehn, "Multiqueue systems with Nonexhaustive Cyclic Service," *The Bell System Tech. Journal*, Vol. 58, 1979.

[7] W. Szpankowski and V. Rego, "Ultimate stability conditions for some multidimensional distributed systems," Purdue CSD-TR 715, October 1987.