Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

ADAPTIVE DISTRIBUTION OF MODEL COMPONENTS VIA CONGESTION MEASURES

Murali S. Shanker
W. David Kelton
Department of Operations and Management Science
Carlson School of Management
University of Minnesota
Minneapolis, MN 55455

Rema Padman
School of Urban and Public Affairs
Carnegie-Mellon University
Pittsburgh, PA 15213

ABSTRACT

One of the factors affecting the performance of distributed simulation models is the assignment of logical processes to processors. This paper outlines a dynamic allocation scheme which can be used for assigning logical processes to processors to minimize the run time of the simulation. The experiments were conducted on an iPSC Hypercube, and indicated that the dynamic scheme reduced and stabilized run times, and was even necessary in some cases for the successful execution of the simulation.

1 INTRODUCTION

With advances in parallel machines, Distributed Simulation (DS) has become a viable way of dealing with time-consuming simulations. An important factor affecting the performance of DS models, like the Chandy and Misra model (Chandy and Misra 1979,1981)(Misra 1986) for DS, is the allocation of logical processes (LPs) to available processors. The objective in making an assignment is to reduce the run time (the actual time taken by the simulation to complete) of the simulation. An "inefficient" assignment may lead to poor performance of the simulation, in some cases worse than that of an equivalent sequential simulation. The problem of assigning LPs to processors to reduce the run time of the simulation is one instance of the *task-allocation* problem found in distributed systems, and it is NP-complete (Garey and Johnson 1979). To distinguish this problem from other task- allocation problems, we will refer to it as the *Distributed Simulation Task Allocation Problem* (DSTAP).

Solution methods for task-allocation problems can be broadly classified into two groups:1) those involving static schemes (Chu and Lan 1987)(Iqbal, Saltz, and Bokhari 1986)(Lee and Aggarwal 1987)(Lo 1988)(Markenscoff and Liaw 1986), and 2) those involving dynamic schemes (Ander 1987)(Iqbal, Saltz, and Bokhari 1986) (Lu and Carey 1986). Static schemes typically perform the allocation of tasks to processors once, generally at the beginning of the simulation, while dynamic schemes try to react to changes in the load on the system, and then make a reallocation. The LPs may be reallocated many times during the length of the simulation using dynamic schemes. Static schemes by their very nature are usually easier and less expensive (in terms of overhead) to use than dynamic schemes. But dynamic strategies are more appropriate for certain situations, especially when the load distribution is likely to change during the simulation.

In DS, the "tasks" are the messages which travel from LP to LP. The path of the message through the logical system depends on the precedence relationship among the LPs, and the path in the computer system depends on the assignment of LPs to the processors. This research will propose a dynamic allocation scheme which can be used for DSTAPs. The proposed dynamic scheme differs in two primary aspects from previous studies:1) in the type of tasks it considers and 2) in the objective it satisfies.

Characteristics of Tasks in DS

In DS, the number of tasks generated is akin to the number of entities generated in a sequential simulation. For a stochastic simulation, the number of entities, and hence the number of tasks, is generally not known beforehand. This uncertainty about the number of tasks precludes the use of schemes which assume that the number of tasks is known. Secondly, the path of the message in the logical system depends on the precedence relationship among the LPs. In most cases, it is not possible to predict the exact path of a message through the logical system as it (the path) is usually updated as the message proceeds from LP to LP. In addition, the amount of work (computation) needed to process a message on an LP (actually on the processor the LP is assigned to—we will say that we process a message on an LP to mean that the message is processed on the processor to which the LP is assigned) depends on the relative position of the message in the system, i.e. it depends on the LP at which the message is located, and the processor to which the LP is assigned. For a fixed LP - Processor assignment, this amount of computation can change during the course of the simulation. This is because

events in a simulation can trigger activities requiring different amounts of work during the simulation. This dynamic nature of the messages will make static schemes (where the assignment is made only once at the beginning of the simulation) inefficient to use for DS problems.

The objective to be Satisfied

The primary objective in making an assignment of LPs to processors is to minimize the run time of the simulation. While experiments have shown (Shaw and Moore 1987) that an unbalanced load can lead to deterioration in run-time performance, achieving a balanced load cannot always be equated to getting the best run-time performance. For example, if communication costs were high, then an unbalanced load over the available processors may lead to better run-time performance than with a balanced load. Load-balancing at best serves as an inadequate objective for minimizing run time.

The rest of the paper is organized as follows: Section 2 describes the problem, and the metric the allocation scheme will use. Section 3 lists the general allocation strategy, and the dynamic allocation scheme, while Section 4 presents the experimental design used to test the schemes. Section 5 contains the results, and the conclusions are in Section 6.

2 THE TASK ALLOCATION PROBLEM

The DSTAP can be stated as follows: Given a logical system with n logical processes (LPs), and a computer system with m processors (P_js) (not necessarily homogeneous), find an assignment ($LP_i \longrightarrow P_j; i = 1,\ldots,n : j = 1,\ldots,m$), static or dynamic, such that the run time of the simulation is minimized.

We use the metric *message-utilization* π_j (Shanker, Kelton, Padman 1989) in developing the dynamic allocation scheme. Specifically, π_j is defined as

$$\pi_j = \lambda_j/\mu_j \; \forall j = 1,\ldots,m$$

where

λ_j = expected arrival rate of *new messages* to processor j

μ_j = expected service rate of messages on processor j

$= 1/\psi_j$

ψ_j = expected service time of messages on processor j

We make the following assumptions in developing a strategy for reallocation:

- We either know or can collect observations to predict expected arrival and expected service rates of messages. No assumption is made about the distributional form of arrivals and service.
- We can collect observations to estimate expected communication rates between two processors for each message type.
- The simulation at any point will behave like its recent past. This is an important assumption because allocation schemes are based on the observations collected, and if the future will be different from what has been observed, then there is no basis for reallocation.

3 STRATEGY FOR REALLOCATION

The dynamic allocation scheme follows a three-phase approach. In the first phase, it identifies the processor(s) in need of reallocation. Next, the LPs which should be reallocated from the affected processor(s) are identified, and finally, the processors to which the LPs are reallocated is decided.

As task reallocation will be performed at execution time, there is a need to keep the allocation scheme simple, but effective. To facilitate this need, two particular strategies will be followed: First, in phase 2, only those LPs which have predecessor or successor LPs on different processors (from the one at which the LP currently resides) will be considered for reallocation. This reduces the number of LPs considered while making a reallocation, and it also helps in maintaining the structure of the logical system, thereby implicitly reducing communication costs. LPs satisfying the above condition will be referred to as *end-LPs*. Secondly, in phase 3, when identifying processors to which to reallocate, those processors containing predecessor or successor LPs of those being reallocated will be chosen first. Such processors will be called *neighbouring processors*. This strategy reinforces the advantages of the earlier strategy in choosing LPs for reallocation.

3.1 The Dynamic Allocation Scheme

In phase 1, a processor is a candidate for reallocation if it becomes *critical*(see Figure 1). For example, a processor j is critical if $\pi_j \geq \pi_k \; \forall k \neq j$, and $\pi_j \geq$ THUL (upper threshold limit set by the modeller). In phase 2, end-Lps in processor j

1. /* Identify processor(s) for reallocation*/
   [Choose processor j such that either
   $\pi_j \geq \pi_k \forall k \neq j$ and π_j >THUL
   or
   $\pi_j \leq \pi_k \forall k \neq j$ and π_j <THLL
   is true]
2. /*Choose LP for reallocation*/
   /*An allocation is needed until $\pi_j \leq$THUL2, or
   $\pi_j \leq 0$ (if originally $\pi_j \leq$THLL)*/
   [While allocation is needed and end-LPs are available DO
   Move end-LP from P_j to available neighbouring
   processor P_k
   Evaluate π_k
   If π_k < THUL2 then
   keep reallocation
   endif
   end DO]

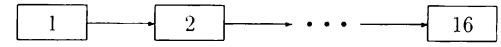
Figure 1: Dynamic Allocation Scheme

are considered for reallocation. End-LPs with predecessor LPs on different processors are considered first, and then those LPs with successor LPs on different processors are considered. In phase 3, the first *available* neighbouring processor is chosen. A processor k is available if $\pi_k \leq$ THUL2 (a parameter set by the modeller). A reallocation is performed by moving an end-LP from the affected processor to a neighbouring processor. The allocation is implemented if the recalculated threshold values of the processors, based on the observed data, remain within threshold limits. If no reallocation is possible the simulation continues with the previous assignment.

A reallocation is performed by shifting load from one processor to another. An alternative way of changing the load in the system is by reducing the rate at which messages are generated. A distributed simulation works as follows: A message is generated at a certain LP called *source-LP*. It is then sent to the next LP in the logical structure, where it is processed and then forwarded to the next LP, etc., until finally the message leaves the system. The rate at which messages are generated depends on the assignment of LPs to processors. If the source-LP is assigned to an overloaded processor, then the rate of message generation will be slow (assuming that a new message is generated only when the previous message has finished its processing on that processor). Alternatively, if the source-LP resides on a relatively free processor, then the rate of message generation will be high. Thus, processors having source-LPs, called *generating processors*, govern the rate of message generation. By changing the load on the generating processors, the load on the system can be changed. This fact coupled with the earlier strategy of reallocating to neighbouring processors provides an effective way of transferring load.

4 EXPERIMENTAL DESIGN

An experimental study was conducted to evaluate the effectiveness of using the dynamic allocation scheme on tasks like those found in distributed simulation. The measure of performance was the run time of the simulation. Two Logical Systems (see Figures 2 and 3) were considered for simulation on an iPSC Hypercube with 4 nodes.



LP_1 = Source-LP
$\boxed{i}$ = LP_i

Figure 2: Logical System 1

The study involved running a distributed simulation of the two logical systems on the Hypercube. The following factors were considered:

A The increase in load. This factor represented the change in processing time for a message on an LP. Initially, a message required equal processing time on all LPs. This factor was considered at two levels.

B The time of load increase. This factor specified the time after which an increase in processing was required. This was considered at two levels.

C The location of load increase. The LP experiencing the load increase is specified here. This also was considered at two levels.

In addition, it was felt that the run length (i.e. the number of messages simulated) of the simulation, and the frequency of reallocation requests, when the scheme was in effect, would influence the run time of the simulation. Unfortunately, due to technological limitations on the buffer space of the Hypercube, the maximum run length that could be simulated was 1500 messages for logical system 1, and 2500 messages for logical system 2. A greater run length resulted in an error while running the simulations, especially when dynamic reallocation was *not* done. The initial conditions, and the design matrix, for the simulations are given in Tables 1 through 4.

All simulations were performed using the Chandy and Misra model for Distributed Simulation (Chandy and Misra 1979,1981), and all data generated were made independent by using nonover-

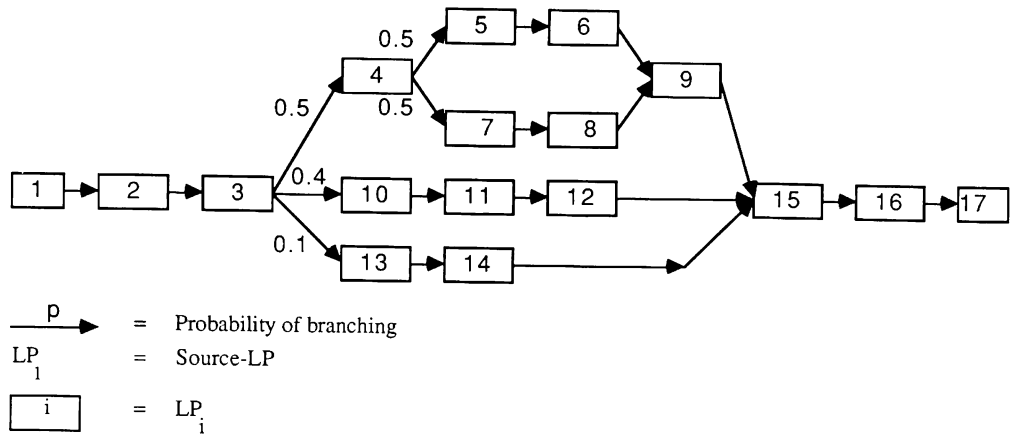


Figure 3: Logical System 2

Table 1: Initial Conditions for Logical System 1

Variable	Value
Run Length	1500 messages
Reallocation Trigger	500 messages
Reallocation Frequency	500 messages
THUL	0.95
THUL2	0.85
THLL	0.20

Table 2: Design Matrix for Logical System 1

Design Point	Factor		
	Load(A)	Time(B)	LP(C)
1	10	200	6
2	20	200	6
3	10	400	6
4	20	400	6
5	10	200	11
6	20	200	11
7	10	400	11
8	20	400	11

Table 3: Initial Conditions for Logical System 2

Variable	Value
Run Length	2500 messages
Reallocation Trigger	800 messages
Reallocation Frequency	800 messages
THUL	0.95
THUL2	0.85
THLL	0.25

Table 4: Design Matrix for Logical System 2

Design Point	Factor		
	Load(A)	Time(B)	LP(C)
1	180	100	5
2	190	100	5
3	180	200	5
4	190	200	5
5	180	100	11
6	190	100	11
7	180	200	11
8	190	200	11

lapping random number streams. The models were simulated using FORTRAN with extended routines for handling node-to-node and node-to-host communications. Thirteen replications of the full design were performed for the first simulation (logical system 1), and 23 for the second simulation (logical system 2). The initial static allocation was chosen so as to achieve the best possible run time for the simulation under the assumption that the initial load remained unchanged for the duration of the simulation.

5 RESULTS

The run time achieved by using the dynamic scheme was at least as good, if not better than, that when running the simulations without the scheme. Figures 4 and 5 show a 90% confidence interval for the average run time for each of the simulations. Though the run times are significantly different only for the first simulation, in both simulations the average run time

is more stable when the scheme is used. This is very apparent for the second simulation (see Figures 3 and 5), where the probabilistic branching of messages at branch points LP_3 and LP_4, coupled with the load increase in these branches, leads to highly variable run times when the scheme is not in use. In contrast, the scheme, by reallocating, is able to stabilize the run time.

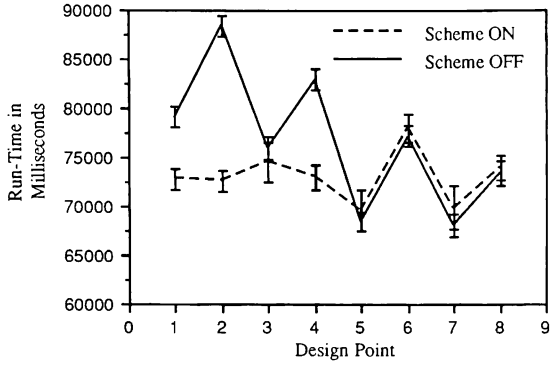


Figure 4: Average Run Time—Logical System 1

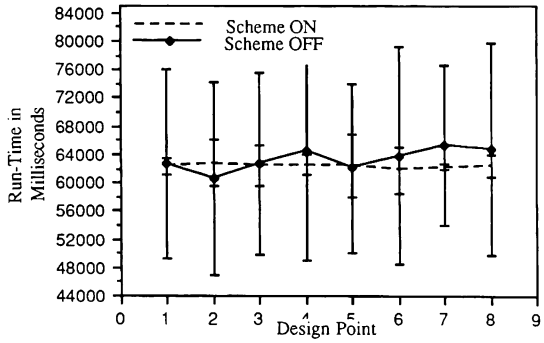


Figure 5: Average Run Time—Logical System 2

In addition, the dynamic scheme appears to use the processors more efficiently. Figures 6 and 7 show the average threshold value achieved across the processors for each design point. The average threshold value is usually lower when the scheme is in use, indicating that the scheme is able to use "less" of the processors to get the same result, thus suggesting that a load increase could be more easily absorbed when the scheme is in use. Also, using the dynamic scheme produced a more balanced load among the processors. Figures 8 and 9 show a 90% confidence interval for the average deviation in load. For logical system 1 there was no statistical difference in the deviation in load (for sake of clarity the confidence limits have not been shown in Figure 8), but for logical system 2, the scheme consistently produced a more balanced load. While it cannot be concluded from these experiments that a balanced load leads to an improvement in

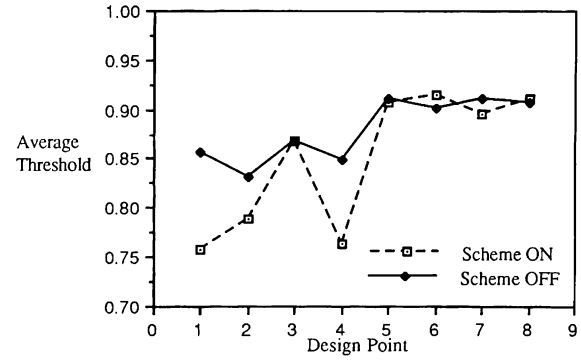


Figure 6: Average Threshold—Logical System 1

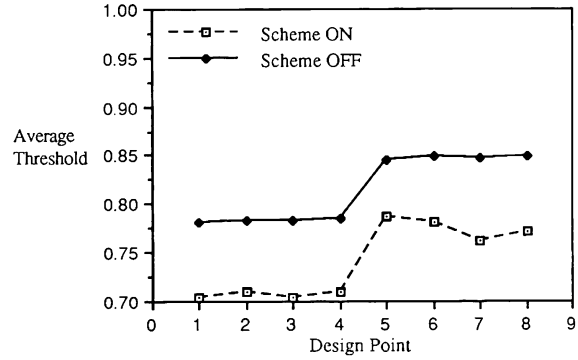


Figure 7: Average Threshold—Logical System 2

run time, earlier simulation experiments did support the theory that a balanced load leads to an improvement in run-time performance, but with one modification. The load is balanced among the processors *used*, and not over all available processors. This is because for certain simulations, communication time becomes an important factor affecting the run time. The number of processors used depends on the assignment made by the scheme in effect. This observation could not be confirmed by the present experiments, as the simulations chosen were processing intensive (each message on an LP took approximately 9.3 milliseconds for the first simulation, and 6.5 milliseconds for the second simulation, before experiencing the load increase), thus reducing the influence of communication time on run time. The simulations were so chosen due to the earlier mentioned limitation of the Hypercube.

To confirm the fact that simulations running with only a static allocation, even a good one, would fail when the run-length increased, logical system 2 was simulated for 5000 messages with all other conditions remaining unchanged. The simulations terminated with an error status (caused by the cube being in a

"hang" status) after approximately 3900 messages were generated. The same simulations completed normally when the dynamic scheme was in effect (see Figure 10), supporting our earlier observation that the scheme uses the processors more efficiently.

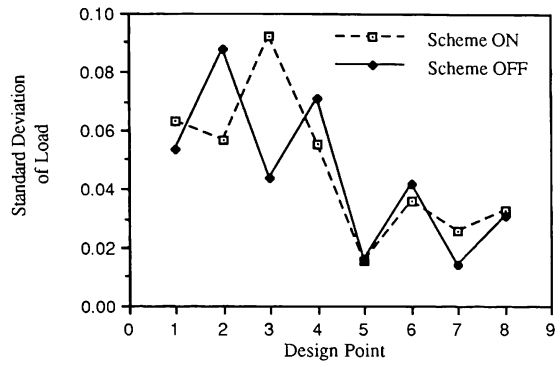


Figure 8: Average Deviation in Load—Logical System 1

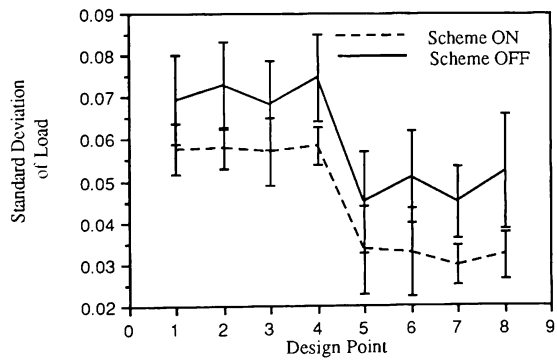


Figure 9: Average Deviation in Load—Logical System 2

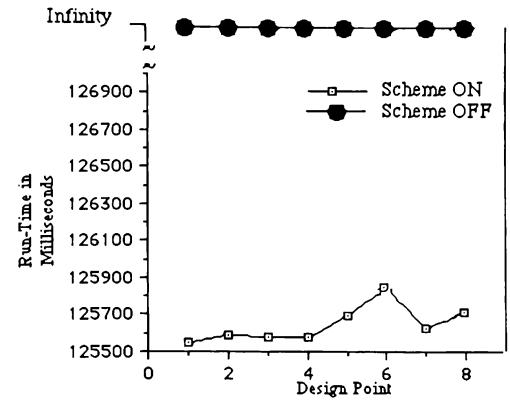


Figure 10: Run Time for 5000 messages—Logical System 2

5.1 Limitations

While using π_j as a metric provides us with the advantage of directly measuring the effect of a reallocation on run time, the calculation of π_j when making a reallocation is based on one important assumption: that the future will behave like the recent past. While the effectiveness of the scheme is not affected for moderate departures from the above assumption, a fact which can be strengthened by choosing appropriate values for THUL and THUL2, it is believed that a major violation of the above assumption could result in a serious miscalculation of π_j, thus resulting in an incorrect reallocation.

In addition, while a Hypercube is a suitable architecture for running distributed simulations, care should be taken in choosing simulations to be run on a machine like the iPSC hypercube. Although most simulations could be run by modifying the synchronization protocols to accommodate the limitations of the cube, this would to a certain extent defeat the purpose of using the particular distributed algorithm.

6 CONCLUSIONS

The dynamic allocation scheme was found to be "cost" effective for running distributed simulations even for very short run-lengths. The experiments resulted in an important conclusion: Dynamic Schemes are needed to produce *feasible* solutions. Also, the initial static allocation was chosen to get favourable run times when the dynamic scheme was not used. In most simulations, this "best" static allocation will generally be not known. In such cases, it will become imperative to use an adaptive

dynamic allocation scheme to run the simulation successfully. While dynamic allocation tends to be expensive (the average time spent on reallocation was approximately 600 milliseconds), the benefits from using a scheme, such as outlined in this paper, far outweigh the costs when applied to tasks like those found in distributed simulations.

ACKNOWLEDGEMENTS

The authors would like to thank Intel Scientific Computers, Beaverton, Oregon, for graciously allowing us to use their Hypercube.

REFERENCES

Andert, E. (1987). A simulation of dynamic task allocation in a distributed computer system. In *Proceedings of the 1987 Winter Simulation Conference*, pp. 768–776.

Chandy, K. M. and J. Misra (1979). Distributed simulation: a case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering, SE-5*, 440–452.

Chandy, K. M. and J. Misra (1981). Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM, 24*, 198–206.

Chu, W. W. and M. Lan (1987). Task allocation and precedence relations for distributed real-time systems. *IEEE Transactions on Computers, C-36*, 667–679.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability*, W. H. Freeman and Company, San Francisco.

Iqbal, A. M., J. H. Saltz, and S. H. Bokhari (1986). A comparative analysis of static and dynamic load balancing strategies. In *Proceedings of the 1986 International Conference on Parallel Processing*, pp. 1040–1047.

Lee, S. and J. K. Aggarwal (1987). A mapping strategy for parallel processing. *IEEE Transactions on Computers, C-36*, 433–442.

Lo, V. M. (1988). Heuristic algorithms for task allocation in distributed systems. *IEEE Transactions on Computers, 37*, 1384–1397.

Lu, H. and M. J. Carey (1986). Load-balanced task allocation in locally distributed computer systems. In *Proceedings of the 1986 International Conference on Parallel Processing*, pp. 1037–1039.

Markenscoff, P. and W. Liaw (1986). Task allocation problems in distributed computer systems. In *Proceedings of the 1986 International Conference on Parallel Processing*, pp. 953–960.

Misra, J. (1986). Distributed discrete-event simulation. *Computing Surveys, 18*, 39–65.

Shanker, M. S., W. D. Kelton, and R. Padman (1989). A metric for dynamic task allocation. *Working Paper, Carlson School of Management, University of Minnesota.*

Shaw, W. H. J. and T. S. Moore (1987). A simulation study of a parallel processor with unbalanced loads. In *Proceedings of the 1987 Winter Simulation Conference*, pp. 759–767.

AUTHORS' BIOGRAPHIES

MURALI S. SHANKER is a Ph.D. candidate in the Carlson School of Management, University of Minnesota. His research interests are in simulation, parallel processing, and in the interface between Computer Science and Management Science. He is a student member of ORSA, TIMS, IEEE, and DSI.

Murali S. Shanker
Department of Operations and Management Science
Carlson School of Management
University of Minnesota
Minneapolis, Minnesota 55455
612/626-7114
murali@vx.acss.umn.edu
murali@umnacvx.bitnet

W. DAVID KELTON is an Associate Professor in the Operations and Management Science Department at the Carlson School of Management, University of Minnesota, in Minneapolis. He received a B.A. in Mathematics from the University of Wisconsin-Madison, an M.S. in Mathematics from Ohio University, and M.S. and Ph.D. degrees in Industrial Engineering from the University of Wisconsin-Madison. His research interests are in simulation methodology, stochastic modeling and estimation, and quality control. He is a member of ORSA, TIMS, and ASA. He served as Program Chair for the 1987 Winter Simulation Conference.

W. David Kelton
Department of Operations and Management Science
Carlson School of Management

University of Minnesota
Minneapolis, Minnesota 55455
612/624-8503
dkelton@vx.acss.umn.edu
dkelton@umnacvx.bitnet

REMA PADMAN is on the faculty of the School of Urban and Public Affairs at Carnegie-Mellon University. Her interests are in parallel and distributed optimization algorithms, scheduling problems, and distributed simulation. Her Ph.D. degree is from the University of Texas at Austin.

Rema Padman
School of Urban and Public Affairs
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
412/268-2159