

Null Message Cancellation in Conservative Distributed Simulation

Bruno R. Preiss, Wayne M. Loucks, Ian D. MacIntyre, James A. Field

Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1

ABSTRACT

This paper presents the results of an empirical study of the effects of null message cancellation on the performance of conservatively synchronized distributed simulation. Null message cancellation is an algorithmic modification to the basic conservative synchronization scheme wherein a null message is discarded before receipt when overcome by a message with a larger timestamp.

Empirical results show that the impact of null message cancellation is affected by the lookahead of the logical process (LP). LPs with high inverse lookahead ratios for both customers and null messages benefit the most from null message cancellation.

Two metrics, weighted fractional decrease in overhead and weighted fractional decrease in messages, are used to assess how much of the “potential” improvement has been captured by null message cancellation.

1. INTRODUCTION

1.1. Background

In [20], Reynolds presents an overview of (distributed) parallel simulation synchronization algorithms. Although, as Reynolds argues, there is a “spectrum” of synchronization algorithms, it is conventional to consider algorithms as being either *optimistic* or *conservative*. Conservative algorithms satisfy the property that no logical process (LP) receives information from another LP that predates the current simulation time of the receiving LP[20]. The basic problems associated with conservative algorithms are to overcome deadlock and to guarantee the steady progress of simulation time.

The seminal work of Chandy and Misra[1], and Peacock, Manning, and Wong[12] avoided deadlock via the use of “null messages” to represent non-events. The problem with null messages is that if their timestamps are generated unintelligently, (e.g., with minimum lookahead, ϵ), the simulation becomes choked with null messages and performance suffers. More intelligent approaches to null-message generation include generation on demand[13], generation after a time-out[3], and the use of “stimulus” nulls[3].

Other recent developments have focussed on incorporating knowledge about the LP into the synchronization algorithms, e.g., Nicol’s “appointments”[10]. Another approach is to constrain the skew in simulation time between different LPs by exploiting knowledge about the LPs and the topology of the interconnect[2, 8].

This paper examines the use of null messages, but incorporates the idea that the timestamps on null messages should be generated intelligently, i.e., by exploiting knowledge about the LP. This is done by explicitly coding the generation and processing of null messages in the model. Thus, the generation of null message timestamps can be done on the basis of the internal state of the LP, rather than simply on the basis of minimum LP delays or topological constraints.

Even intelligent generation of null message timestamps can lead to poor performance due to choking. In this paper, the impact of the cancellation of spare null messages is examined. The following section describes how null message cancellation is done.

1.2. Null Message Cancellation

Consider two logical processes, A and B. If A sends a null message with timestamp t to B at real time τ , it can be viewed as a promise by A that it will not send another message (null or non-null) with timestamp t' , $t' \leq t$ at any real time after τ . Suppose A later sends another message (null or non-null) with timestamp t'' . (N.B., t'' must be greater than t .) Furthermore, suppose that the second message arrives at B before B has processed the first (null) message. Then, the first (null) message can be discarded. This is called null message cancellation (or

annihilation)[9].

In the example discussed above, the null message was cancelled at the destination. In fact, if a message ever “catches up” to a null message, the (earlier) null message can be discarded. I.e., null message cancellation can be done en-route. This might be the case in a store-and-forward message passing network with queuing at each intermediate node. In this paper, the impact of null message cancellation at the destination on the performance of parallel simulation is examined.

2. EXPERIMENTAL OVERVIEW

2.1. Benchmarks

The results presented in this paper were obtained from parallel simulations of closed stochastic queuing network benchmarks based on those described in [11]. The following is a very brief description of the benchmarks. For more detailed descriptions see [7, 15, 17, 18].

The system simulated is a static network of nodes. Each node has f_{in} inputs, f_{out} outputs, a single queue, and a single server. The queuing discipline is FCFS. Service is non-preemptive. The service time distribution is a biased, exponentially distributed random variable. The minimum service time is μ_{min} ; the mean service time is $\mu = 5\mu_{min}$; and, the system was simulated for 1000μ time steps.

The suite of benchmarks includes four network topologies — ring ($f_{in}=f_{out}=2$), multistage ($f_{in}=f_{out}=2$), mesh ($f_{in}=f_{out}=4$), hypercube ($f_{in}=f_{out}=6$). In this paper, only mesh results are presented. The results for the other topologies are qualitatively similar. The effects of topology on the results will be mentioned in the summary. In all cases, the system consists of 64 nodes.

The number of customers in the system is a measure of the simulation “load”. The time-averaged number of customers in each node is given by N . Simulations were done with $N=1$, $N=4$, and $N=8$.

Each node in the network is modelled by a single logical process (LP). Four different implementations of the LP have been studied. Each LP implementation provides a different degree of “lookahead” (defined below) by taking advantage of certain aspects of the system being simulated.

2.2. Simulation Software and Hardware

The simulations were implemented using the Yaddes distributed discrete event simulation language[14, 16, 19]. Two variants of the parallel simulation kernel using the conservative synchronization protocol were used — *with* and *without* null message cancellation.

The simulations were performed on a Transputer multiprocessor with eight processors. The connections between the processors form a cube. Speedup is measured with respect to a single Transputer processor running a conventional sequential (event-list-driven) discrete event simulation kernel.

3. CHARACTERIZING THE LOOKAHEAD OF LOGICAL PROCESSES

The concepts of lookahead and inverse lookahead ratio are important in understanding the performance of parallel simulations. In this section, these concepts are reviewed and a metric, introduced in [17], called *null message lookahead* is defined.

3.1. Characteristic Times of Messages vs. Null Messages

Figure 1 shows the evolution of the simulation time of an LP as a function of real time. Consider the activity associated with simulating the servicing of the i^{th} customer. In the system being simulated, customer i arrives at the node at simulation

time t_{cause} . The processing of the event representing the arrival of this customer begins at real time τ_0 . In the system being simulated, customer i departs from the node at simulation time t_{effect}^{cust} . The processing of the event representing the departure of the customer ends at real time τ_3 . Thus, during the real time interval between τ_0 and τ_3 (i.e., the customer i processing window), an irrevocable decision (i.e., that customer i will depart at simulation time $t_{effect}^{cust \dagger}$).

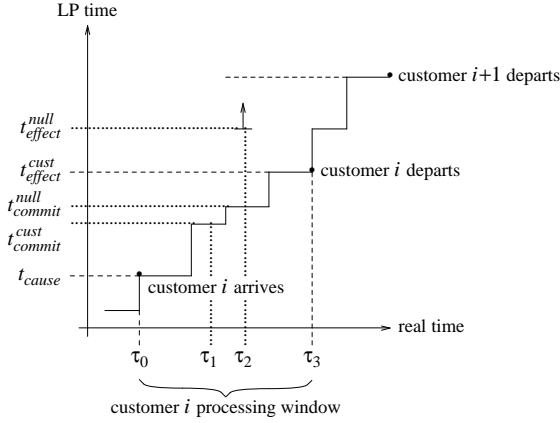


Figure 1. LP Time vs. Real Time

Assume that at some real time τ_1 during the processing window for the i^{th} customer, the LP determines with certainty that the departure time of customer i will be t_{effect}^{cust} . Furthermore, assume that the LP determines that it can safely transmit at real time τ_1 an output message representing the departure of customer i , i.e., a customer message with timestamp t_{effect}^{cust} . Note, in order for it to be safe to transmit the message at real time τ_1 it must be the case that no message will be sent by this LP (via the same output) with timestamp t , $t < t_{effect}^{cust}$ at any real time τ , $\tau > \tau_1$.

In effect, at real time τ_1 , the LP commits to the fact that customer i will depart at time t_{effect}^{cust} . The value of LP time at the point when the commitment is made is called t_{commit}^{cust} .

In addition to sending customer-bearing messages, an LP must send null messages in order to avoid deadlock. A null message with timestamp t is a promise that no customer-bearing message will be sent on the same link with timestamp t' , $t' \leq t$. Thus, it is safe to advance the link time to time t . In effect, null messages represent the “non-departure” of the next customer.

Figure 1 shows the departure of the $(i+1)^{th}$ customer from the given LP. Assume that at some real time τ_2 , the LP determines with certainty that the departure time of customer $i+1$ will be at some LP time t , $t > t_{effect}^{null}$. Furthermore, assume that the LP determines that it can safely transmit at real time τ_2 a null message representing the “non-departure” of customer $i+1$, i.e., a null message with timestamp t_{effect}^{null} . Note, in order for it to be safe to transmit the message at real time τ_2 it must be the case that a message will not be sent by this LP (via the same output) with timestamp t , $t < t_{effect}^{null}$, at any real time τ , $\tau > \tau_2$.

In effect, at real time τ_2 , the LP commits to the fact that customer $i+1$ will depart at some LP time after t_{effect}^{null} . The value of LP time at the point when the commitment is made is called t_{commit}^{null} .

\dagger In fact, if it is possible to predict with certainty the arrival time of customer i at some real time τ such that $\tau < \tau_0$, then it may also be possible to predict the departure time of customer i at some other real time τ' such that $\tau \leq \tau' \leq \tau_0$. However, this situation is not likely in typical simulations because it requires the given LP to “know” something about the behaviour and current state of other LPs and about the topology of the interconnection of the LPs.

Tables I and II summarize the characteristic customer and null message times for the four different LP implementations. Table I gives the characteristic customer and null message times for the case where a customer arrives at an LP at simulation time t_{cause} and departs at simulation time $t_{effect}^{cust} = t_{cause} + t_{queueing} + t_{service}$. Table II summarizes the characteristic null message times in the case where only null messages have been received by the LP at simulation time t_{cause} . In this case, since no customer has arrived, no customer will depart. However, null messages will be sent. Thus, Table II only gives values for t_{commit}^{null} and t_{effect}^{null} .

Table I
Characteristic Message Times Upon Customer Arrival

model	t_{effect}^{null}	t_{commit}^{cust} , t_{commit}^{null}
epsilon	t_{effect}^{cust}	$t_{cause} + t_{queueing} + t_{service} - \epsilon$
service time	$t_{effect}^{cust} + \mu_{min}$	$t_{cause} + t_{queueing}$
system time	$t_{effect}^{cust} + \mu_{min}$	t_{cause}
system time + presampling	$t_{effect}^{cust} + t_{prediction}$	t_{cause}

Table II
Characteristic Message Times Upon Null Message Arrival

model	t_{effect}^{null}	t_{commit}^{null}
epsilon	$t_{cause} + 2\epsilon$	t_{cause}
service time	$t_{cause} + \mu_{min}$	t_{cause}
system time	$t_{cause} + \mu_{min}$	t_{cause}
system time + presampling	$t_{cause} + t_{prediction}$	t_{cause}

The *epsilon* model commits to the departure of a customer ϵ (in LP time) before the customer actually departs. The *service time* model takes advantage of the fact that service is non-preemptive and commits to the departure of a customer $t_{service}$ before the the customer departs. The *system time* model takes advantage of the fact that the queueing discipline is FCFS and commits to the departure of a customer $t_{queueing} + t_{service}$ before the customer departs.

In all the models used in this study, $t_{commit}^{null} = t_{commit}^{cust}$ for null messages sent in conjunction with customer arrivals, as shown in Table I. The *epsilon* model sends null messages with the same timestamps as customer messages on all the outputs of the LP except the one from which the customer departs. The *service time* and *system time* models take advantage of the fact that the service time distribution is biased (i.e., it has a minimum, μ_{min}) and send null messages with timestamps μ_{min} greater than the customer departure time.

The *system time + presampling* model is the same as the *system time* model with respect to customer departures. However, this model takes advantage of the fact that customers are indistinguishable and presamples the random number generator to “predict” the earliest possible departure times of subsequent customers. This is done

using a futurelist[11].

3.2. Lookahead vs. Null Message Lookahead

In this section, a measure of the predictiveness of an LP called “lookahead” is defined. For a survey of the various definitions of lookahead used in the literature, see also [17]. The key idea here is the observation that null messages carry predictions and therefore have “lookahead” just as customer-bearing messages do.

Definition: The **(customer) lookahead** of an LP (with respect to customer i) is the quantity $t_{effect}^{cust} - t_{commit}^{cust}$.

Lookahead is a measure of the “predictiveness” of an LP in that it measures at real time τ_1 the difference between the local simulation time of the LP, t_{commit}^{cust} , and the timestamp, t_{effect}^{cust} , on the outgoing message generated by that LP at that real time. It is generally accepted that parallel simulations consisting of LPs having large lookahead will exhibit better speedup[5,6]. Note that the association between lookahead and speedup is indirect in that lookahead measures a quantity in simulation time, whereas speedup is measured in real time.

Definition: The **null message lookahead** of an LP (with respect to customer i) is the quantity $t_{effect}^{null} - t_{commit}^{null}$.

Null message lookahead, like (customer) lookahead is another measure of the “predictiveness” of an LP. Null message lookahead measures at real time τ_2 the difference between the local simulation time of the LP t_{commit}^{null} and the timestamp t_{effect}^{null} on the outgoing null message generated by that LP at that real time.

Table III shows the values for (customer) lookahead and null message lookahead for the four different LP implementations. This table illustrates that there are many other possible LP implementations. The LPs selected in this study have comparable values for customer and null message lookahead.

Table III
Lookahead Models

null message lookahead	customer lookahead		
	ϵ	$t_{service}$	$t_{queueing} + t_{service}$
ϵ	epsilon	–	–
$t_{service} + \mu_{min}$	–	service time	–
$t_{queueing} + t_{service} + \mu_{min}$	–	–	system time
$t_{queueing} + t_{service} + t_{prediction}$	–	–	system time + presampling

3.3. ILAR vs. NILAR

It is well known that it is not the absolute value of lookahead that is important, but rather its value relative to the “average timestamp increment”[4]. The inverse lookahead ratio (ILAR) for an LP is given by

$$ILAR = \frac{E[t_{effect}^{cust} - t_{commit}^{cust}]}{E[t_{effect}^{cust} - t_{cause}]}, \quad (1)$$

where the expectation is taken over all customers.

Similar arguments can be made about null message lookahead[17]. The null message inverse lookahead ratio (NILAR) for an LP is given by

$$NILAR = \frac{E[t_{effect}^{null} - t_{commit}^{null}]}{E[t_{effect}^{null} - t_{cause}]}, \quad (2)$$

where the expectation is taken over all null messages *sent in association with customers*. (Note the use of t_{effect}^{cust} in the denominator of NILAR.)

Table IV shows expressions for ILAR and NILAR.

4. PERFORMANCE METRICS

Table IV

model	ILAR	NILAR
epsilon	$\frac{\epsilon/\mu}{N}$	$\frac{2\epsilon/\mu}{N}$
service time	$\frac{1}{N}$	$\frac{1 + \mu_{min}/\mu}{N}$
system time	1	$\frac{N + \mu_{min}/\mu}{N}$
system time + presampling	1	$\frac{N + f_{out}}{N}$

In this section, two metrics used to evaluate the benefit of null message cancellation are defined. They are *weighted fractional decrease in overhead* and *weighted fractional decrease in messages*. These metrics are used in order to fairly assess the benefit due to null message cancellation. The situation to avoid is best illustrated by an example. Suppose the parallel efficiency of simulation without null message cancellation is 0.01 and with null message cancellation is 0.05. Then it could be said that null message cancellation has improved performance by a factor of 5. However, the resulting simulation is still terribly inefficient. What is needed are metrics that show how much of the potential improvement was captured by null message cancellation.

4.1. Weighted Fractional Decrease in Overhead

This metric measures the improvement caused by null message cancellation in the amount of time lost due to the overhead associated with parallel execution. Define the parallel execution “overhead” as the difference between the time to execute the simulation on a multiprocessor with n processors and the “ideal” multiprocessor execution time (computed by dividing the uniprocessor execution time by n). Note, this definition lumps processor idle time together with processor time spent doing message passing and synchronization. Let T_n^{all} be the time to execute the simulation on n processors *without* null message cancellation (i.e., *all* null messages are processed). Let T_1 be the uniprocessor execution time of the same simulation. Then the overhead associated with the parallel execution of the simulation without null message cancellation is $O^{all} = T_n^{all} - T_1/n$. Similarly, the overhead *with* null message cancellation is $O^{cancel} = T_n^{cancel} - T_1/n$. The fractional decrease in overhead due to null message cancellation is given by

$$\frac{\Delta O}{O^{all}} = \frac{O^{all} - O^{cancel}}{O^{all}} = \frac{T_n^{all} - T_n^{cancel}}{T_n^{all} - T_1/n}. \quad (3)$$

Note, a large fractional decrease in overhead is not necessarily associated with good speedup. In order to identify the cases where the improvement due to null message cancellation results in good final speedup, the fractional decrease in overhead is multiplied (weighted) by the parallel efficiency of the simulation done with cancellation. Parallel efficiency, η , is the “ideal” multiprocessor execution time defined as above divided by the time to execute the simulation on a multiprocessor with n processors. Thus, the *weighted* fractional decrease in overhead (WFDIO) is given by

$$WFDIO = \frac{T_n^{all} - T_n^{cancel}}{T_n^{all} - T_1/n} \eta^{cancel} = \frac{\eta^{cancel} - \eta^{all}}{1 - \eta^{all}}, \quad (4)$$

where $\eta^{cancel} = T_1/n T_n^{cancel}$, and $\eta^{all} = T_1/n T_n^{all}$,

Note, as shown above, WFDIO can also be interpreted as the change in efficiency due to null message cancellation, $\eta^{cancel} - \eta^{all}$, divided by the *potential* improvement in efficiency, $1 - \eta^{all}$. Thus, WFDIO measures the fraction of the potential improvement in performance that has been achieved by null message cancellation.

4.2. Weighted Fractional Decrease in (Null) Messages

The weighted fractional decrease in messages (WFDIM) measures the effect of null message cancellation on the number of null messages required to perform the parallel simulation. The definition of WFDIM is analogous to that of WFDIO (Equation 4). Recall, WFDIO is the ratio of actual improvement to potential improvement weighted by the resulting parallel efficiency. WFDIM is the ratio of the *difference* in

the number of null messages between parallel simulation with and without null message cancellation (actual improvement) to the number of null messages required without null message cancellation (potential improvement) weighted by the resulting parallel efficiency. I.e.,

$$WFDIM = \frac{M_n^{all} - M_n^{cancel}}{M_n^{all} - 0} \eta^{cancel}, \quad (5)$$

where M_n^{all} is the number of null messages required in the parallel simulation without null message cancellation, and M_n^{cancel} is the number of null messages required in the parallel simulation with null message cancellation.

5. OBSERVATIONS

Figure 2 shows the speedup S_n^{all} , $S_n^{all} = T_1/T_n^{all}$, without null message cancellation vs. the number of processors, n . Figure 3 shows the speedup S_n^{cancel} , $S_n^{cancel} = T_1/T_n^{cancel}$, with null message cancellation vs. the number of processors. Figures 2 and 3 show results for the mesh topology benchmark under various loads and using various lookahead models. Note that reasonable speedup can be achieved under moderate to heavy load using the *system-time* and *system-time + presampling* lookahead models. Note also that under lighter loads and with less predictive lookahead models, substantial “slowdown” occurs. I.e., the parallel simulation on n processors runs more slowly than the sequential simulation. The effectiveness of null message cancellation is demonstrated by the change in speedup between Figures 2 and 3. However, simply taking the ratio of speedups can be misleading.

Figure 4 relates the performance of the simulations in Figures 2 and 3. It is a scatter plot of parallel efficiency with cancellation ($\eta^{cancel} = S_n^{cancel}/n$) vs. efficiency without cancellation ($\eta^{all} = S_n^{all}/n$). In this figure, points lying near the diagonal have benefited the least from null message cancellation, whereas points near the horizontal line at $\eta^{cancel} = 1$ have benefited the most. The cluster of points near the diagonal correspond to the simulations exhibiting “slowdown”. This figure clearly illustrates the fallacy in taking the ratio of speedup (or efficiency) with and without cancellation. E.g., null message cancellation can increase parallel efficiency by a factor of four. However, the resulting simulation may still be very inefficient!

Figure 5 shows WFDIO vs. NILAR for the simulation data obtained using $n=8$ processors. (The data for other values of n is qualitatively similar.) This figure clearly shows the following points: 1. As NILAR increases (specifically, NILAR greater than one), the impact of null message cancellation also increases. 2. As load increases, increasing NILAR has a stronger impact on WFDIO. Thus, systems with large NILAR and large load are likely to reap the most benefit from null message cancellation.

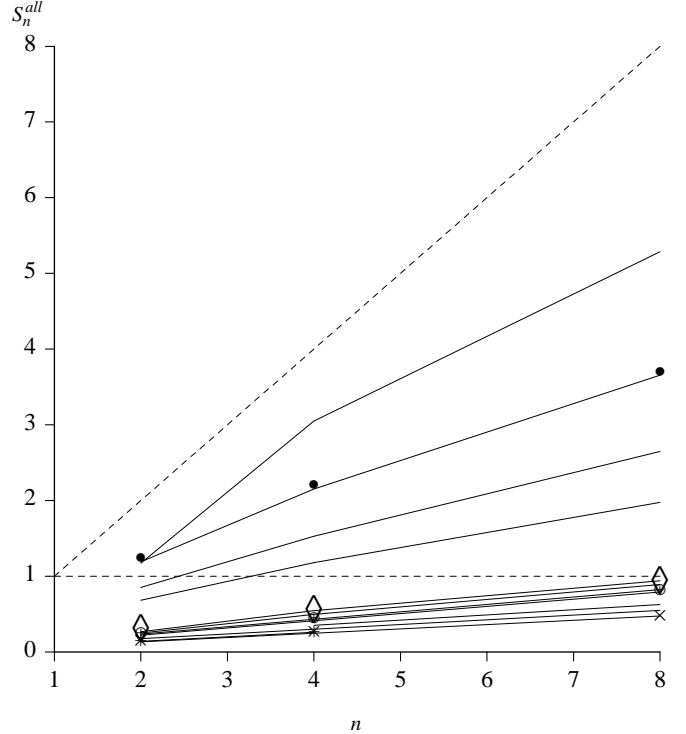
Figure 5 clearly shows that ILAR by itself is not a very good predictor of the benefit of null message cancellation. Specifically: 1. Null message cancellation has little effect for ILAR less than one. 2. ILAR equal to one does not guarantee that null message cancellation will be beneficial.

Figure 6 shows WFDIM vs. NILAR for the simulation data obtained using $n=8$ processors. This figure indicates that null message cancellation affects the number of null messages in the same way as (time) overhead with respect to NILAR and load.

5.1. Summary of other observations

This section is a brief summary of other observations regarding the impact of null message cancellation. (Unfortunately, space restrictions prevent the inclusions of raw data.)

- The effect of null message cancellation does not depend on the number of processors, n , used to perform the parallel simulation.
- Using the *system time + presampling* model, the improvement in performance due to null message cancellation increases as the fanout, f_{out} , of the LP increases. I.e., the benefit due to null message cancellation was largest for the hypercube topology and smallest for the ring topology. This effect is predicted by the fact that NILAR increases with increasing f_{out} and by the correlation between NILAR and WFDIO.



Legend:

- +—+ load=1, Epsilon
- ×—× load=1, Service-time
- load=1, System-time
- load=1, System-time + Presampling
- ⊕—⊕ load=4, Epsilon
- load=4, Service-time
- load=4, System-time
- ◇—◇ load=4, System-time + Presampling
- ◇—◇ load=8, Epsilon
- load=8, Service-time
- load=8, System-time
- load=8, System-time + Presampling

Figure 2. Speedup vs. Processors (no cancellation).

- Null message cancellation can result in the elimination of up to 50% of the messages with no change in speedup. I.e., null message cancellation can cause processors doing “busy work” to become idle. However, neither condition contributes to speedup.
- Null message cancellation can improve speedup by as much as a factor of three with close to 75% of messages being cancelled (e.g., system-time + presampling lookahead model, load=1, eight processors). However, null message cancellation cannot save a poor parallel simulation.

6. SUMMARY

This paper examines the impact of null message cancellation on the performance of conservative distributed simulation. The benchmarks used are simulations of closed, stochastic queueing networks with various topologies. Four different LP implementations are examined, each with a different degree of predictiveness. The predictiveness of each LP is characterized by two quantities — (customer) inverse lookahead ratio (ILAR) and null message inverse lookahead ratio (NILAR). These quantities measure the relative lookahead of an LP with respect to customer-bearing messages and null messages (respectively).

Two metrics are used to assess the impact of null message cancellation on the performance of the simulation. Weighted fractional decrease in overhead (WFDIO) is shown to be the fraction of the potential increase in parallel efficiency

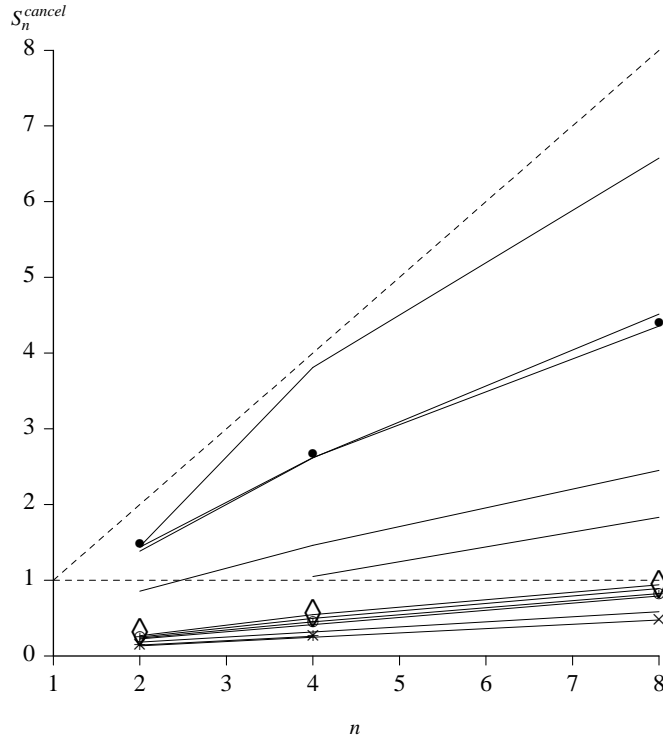


Figure 3. Speedup vs. Processors (with cancellation).
(Legend as in Figure 2.)

captured by null message cancellation. The results show that for those LPs with an ILAR of one the benefit due to null message cancellation increases with increasing NILAR. Similar results are shown for the weighted fractional decrease in messages (WFDIM).

On the basis of this study, it is concluded that null message cancellation is a viable algorithmic modification to the conventional conservative synchronization method when coupled with an appropriately predictive LP implementation. Furthermore, ILAR alone is not a good indicator of the predictiveness of an LP. On the contrary, the results indicate that the more predictive an LP is with respect to null messages, the more likely it is to benefit from null message cancellation!

7. ACKNOWLEDGEMENTS

This work was supported in part by the Information Technology Research Centre (ITRC) of the Province of Ontario (Canada) and by the Natural Sciences and Engineering Research Council (NSERC) of Canada under grants OGP6840, OGP36635 and EQ43585.

8. REFERENCES

1. Chandy, K. M. and Misra, J., "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Trans. on Software Engineering*, Vol. SE-5, No. 5, pp. 440-452, September 1979.
2. Cota, B. A. and Sargent, R. G., "An Algorithm for Parallel Discrete Event Simulation using Common Memory," *Proc. 22nd Ann. Simulation Symp.*, pp. 23-31, March 1989.
3. Davis, N. J., Mannix, D. L., Shaw, W. H., and Hartrum, T. C., "Distributed Discrete-Event Simulation Using Null Message Algorithms on Hypercube Architectures," *Journal of Parallel and Distributed Computing*, Vol. 8, No. 4, pp. 349-357, April 1990.
4. Fujimoto, R. M., "Performance Measurements of Distributed Simulation Strategies," Technical Report No. UUCS-87-026a, Computer Science Department, University of Utah, Salt Lake City, Utah, 1987.

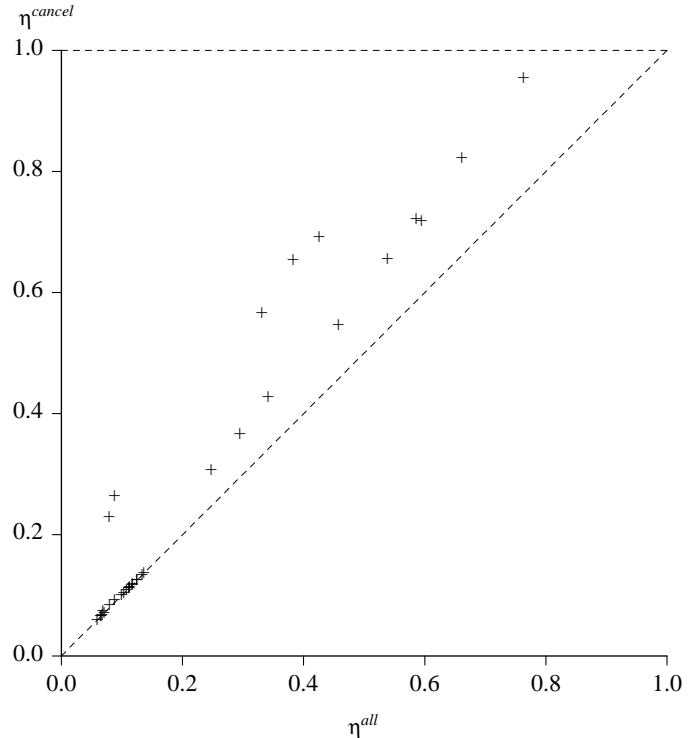


Figure 4. Efficiency vs. Efficiency.

5. Fujimoto, R. M., "Lookahead in Parallel Discrete Event Simulation," *Proc. 1988 Int. Conf. on Parallel Processing*, Vol. III, pp. 34-41, August 1988.
6. Fujimoto, R. M., "Performance Measurements of Distributed Simulation Strategies," *Trans. of the Society for Computer Simulation*, Vol. 6, No. 2, pp. 89-132, April 1989.
7. Loucks, W. M. and Preiss, B. R., "The Role of Knowledge in Distributed Simulation," *Proc. SCS Eastern Multiconf. — Distributed Simulation*, pp. 9-16, Society for Computer Simulation, January 1990.
8. Lubachevsky, B. D., "Bounded Lag Distributed Discrete Event Simulation," *Proc. SCS Eastern Multiconf. — Distributed Simulation*, Vol. 19, No. 3, pp. 183-191, Society for Computer Simulation, February 1988.
9. Misra, J., "Distributed Discrete-Event Simulation," *ACM Computing Surveys*, Vol. 18, No. 1, pp. 39-66, March 1986.
10. Nicol, D. M. and Reynolds, P. F., "Problem Oriented Protocol Design," *Proc. 1984 Winter Simulation Conf.*, pp. 471-474, Nov. 1984.
11. Nicol, D. M., "High Performance Parallelized Discrete Event Simulation of Stochastic Queueing Networks," *Proc. 1988 Winter Simulation Conf.*, pp. 306-314, Society for Computer Simulation, December 1988.
12. Peacock, J. K., Wong, J. W., and Manning, E., "A Distributed Approach to Queueing Network Simulation," *Proc. 1979 Winter Simulation Conf.*, pp. 399-406, Dec. 1979.
13. Peacock, J. K., Wong, J. W., and Manning, E., "Synchronization of Distributed Simulation Using Broadcast Algorithms," *Computer Networks*, Vol. 4, pp. 3-10, 1980.
14. Preiss, B. R., Loucks, W. M., and Hamacher, V. C., "A Unified Modeling Methodology for Performance Evaluation of Distributed Discrete Event Simulation Mechanisms," *Proc. 1988 Winter Simulation Conf.*, pp. 315-324, Society for Computer Simulation, December 1988.

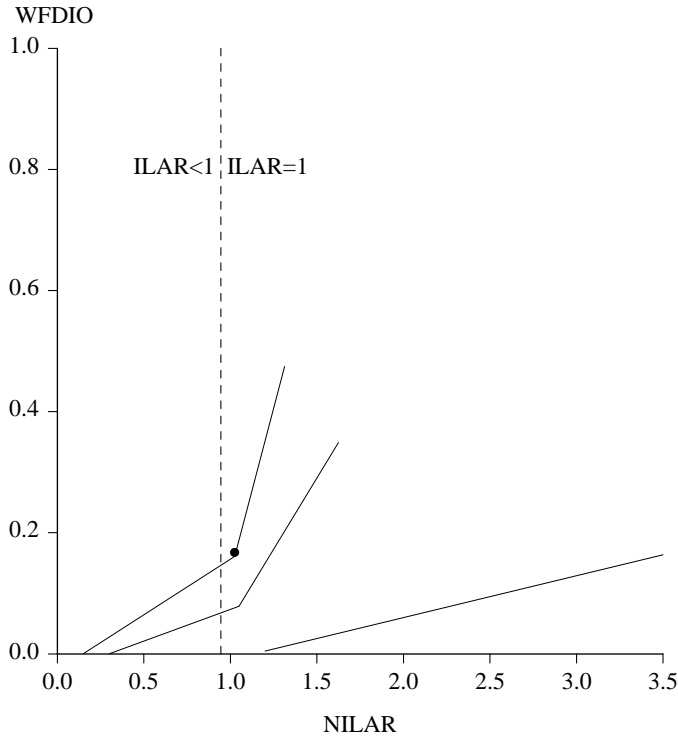


Figure 5. WFDIO vs. NILAR, processors=8.
(Legend as in Figure 6.)

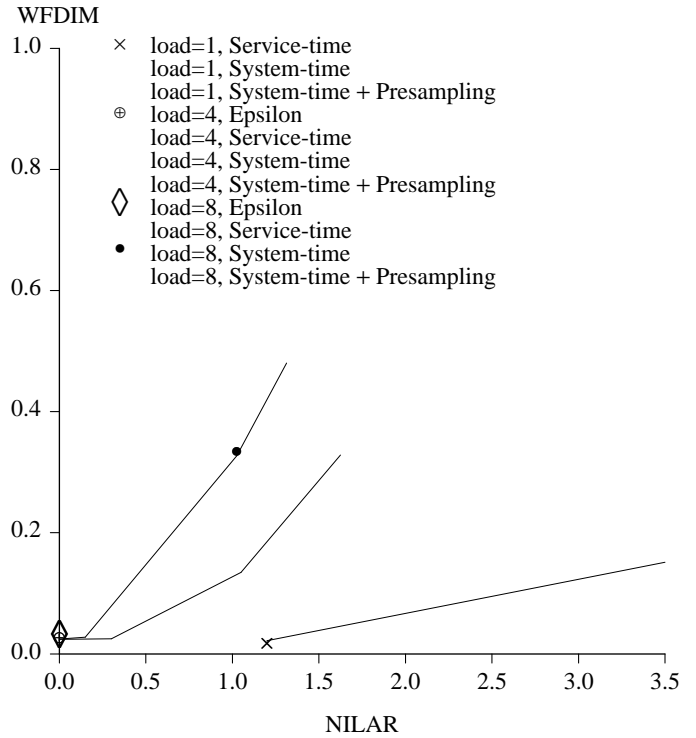


Figure 6. WFDIM vs. NILAR, processors=8.

15. Preiss, B. R. and Loucks, W. M., "Prediction and Lookahead in Distributed Simulation," CCNG Technical Report E-191, Computer Communications Networks Group, University of Waterloo, Waterloo, Ontario, Canada, 1989.
16. Preiss, B. R., "The Yaddes Distributed Discrete Event Simulation Specification Language and Execution Environments," *Proc. SCS Eastern Multiconf. — Distributed Simulation*, Vol. 21, No. 2, pp. 139-144, Society for Computer Simulation, March 1989.
17. Preiss, B. R. and Loucks, W. M., "The Impact of Lookahead on the Performance of Conservative Distributed Simulation," *Proc. SCS European Multiconf. — Simulation Methodologies, Languages and Architectures*, pp. 204-209, Society for Computer Simulation, June 1990.
18. Preiss, B. R., "Performance of Discrete Event Simulation on a Multiprocessor Using Optimistic and Conservative Synchronization," *Proc. 1990 Int. Conf. on Parallel Processing*, Vol. III, pp. 218-222, August 1990.
19. Preiss, B. R. and MacIntyre, I. D., "YADDES — Yet Another Distributed Discrete Event Simulator: User Manual," CCNG Technical Report E-197, Computer Communications Networks Group, University of Waterloo, Waterloo, Ontario, Canada, 1990.
20. Reynolds, P. F., "A Spectrum of Options for Parallel Simulation," *Proc. 1988 Winter Simulation Conf.*, pp. 325-332, Society for Computer Simulation, December 1988.