The Impact of Lookahead on the Performance of Conservative Distributed Simulation

Bruno R. Preiss

Wayne M. Loucks

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

ABSTRACT

It has long been recognized that the ability of a logical process (LP) to “look ahead” is crucial to the performance of distributed simulation using conservative synchronization. This paper examines previous definitions of lookahead and lookahead ratio — metrics for characterizing the lookahead ability of LPs.

These metrics do not reflect the use of null messages. In order to assess the impact of null messages on the performance of the simulation, we introduce the concept of null message lookahead and null message inverse lookahead ratio (NILAR). In this paper, we also propose a consistent set of definitions for lookahead and inverse lookahead ratio (ILAR) for customer-neutral systems.

The results discussed in this paper indicate that NILAR acts as a better predictor of simulation speedup than ILAR in the case of closed queueing networks with a large number of null messages. The results presented also indicate that conservatively synchronized distributed simulation implementations must have ILAR values close to one and NILAR values greater than one in order to achieve good speedup. However, ILAR values close to one and NILAR values greater than one do not necessarily result in good speedups. Other requirements are higher loads and limited fanout.

1. INTRODUCTION

In this study, we examine the performance of distributed discrete event simulations of closed, stochastic queueing networks. We examine the same suite of four networks as in (Nicol 1988). (See also (Loucks and Preiss 1990).) In each case, the system simulated is a static network of nodes. Each node has $n$ inputs, a queue, a server, and $n$ outputs. Various network topologies and loads are considered.

The simulation specifications are coded in the Yaddes language (Preiss 1989). The simulations were executed on a set of eight Transputers and the speedup was calculated relative to a single event-list version running on a single processor. Although our implementation of Yaddes supports three different distributed simulation synchronization methods, in this paper we consider only the conservative Chandy-Misra (Chandy and Misra 1979, Misra 1986) (with null messages) technique.

Because these networks contain many instances of the same type of node, only one model specification is required. We present four different implementations of the node model specification — each with a varying degree of lookahead (see below). It is important to emphasize that, although the implementations have different lookaheads, the physical systems simulated are identical (for a given set of initial conditions and topology).

Distributed simulation holds the promise of speedup. However, results from some tests have been mixed. In this paper we emphasize the use of lookahead (and some improved versions of lookahead) as an indicator for possible performance improvement.

In Section 2 we examine several traditional views of “lookahead” and propose a new measure (null message lookahead) based on the predictive nature of the null messages in the system. In Section 3 the experimental method and test suite are described. Section 4 describes our results and finally, Section 5 discusses the impact of this work.

2. LOOKAHEAD AS A MEASURE OF SIMULATION IMPLEMENTATION EFFECTIVENESS

Since lookahead was first introduced by Chandy and Misra (Chandy and Misra 1979) it has taken on a number of (slightly) different meanings and interpretations. In this section we put lookahead (and its partner lookahead ratio) into the context of this paper. Prior to discussing lookahead as a topic, it is convenient to examine a simple example as a basis for discussion.

2.1. A Simple Queueing Example

Consider a system consisting of a single queue and server as shown in Figure 1. In this system customers arrive on the input arc, enter a first-come, first-served (FCFS) queue, and are eventually served by the server. The service time is a biased, exponentially distributed random variable with a mean of $\mu$ and minimum of $\mu_{\text{min}}$. Service is nonpreemptive.

Distributed simulation holds the promise of speedup. However, results from some tests have been mixed. In this paper we emphasize the use of lookahead (and some improved versions of lookahead) as an indicator for possible performance improvement.

In Section 2 we examine several traditional views of “lookahead” and propose a new measure (null message lookahead) based on the predictive nature of the null messages in the system. In Section 3 the experimental method and test suite are described. Section 4 describes our results and finally, Section 5 discusses the impact of this work.

2. LOOKAHEAD AS A MEASURE OF SIMULATION IMPLEMENTATION EFFECTIVENESS

Since lookahead was first introduced by Chandy and Misra (Chandy and Misra 1979) it has taken on a number of (slightly) different meanings and interpretations. In this section we put lookahead (and its partner lookahead ratio) into the context of this paper. Prior to discussing lookahead as a topic, it is convenient to examine a simple example as a basis for discussion.

2.1. A Simple Queueing Example

Consider a system consisting of a single queue and server as shown in Figure 1. In this system customers arrive on the input arc, enter a first-come, first-served (FCFS) queue, and are eventually served by the server. The service time is a biased, exponentially distributed random variable with a mean of $\mu$ and minimum of $\mu_{\text{min}}$. Service is nonpreemptive.

Consider the following scenario. At time 30 customer A starts service (with a total service time of 12) and customers B, C, and D are in the queue awaiting service. When they are served, their service times will be 3, 5, and 11, respectively. At time 32 a new customer, E, arrives. Its service time will be 9. Table 1 gives the start and end times of the service for each customer in the system.

![Figure 1. Single FCFS Queueing System](image_url)
The issues associated with lookahead can be described in terms of the behaviour of the logical process (LP) simulating the queueing system as it evaluates customer E. The LP which simulates the queue shown in Figure 1 is shown in Figure 2. Once the LP has completed executing with a simulation time of 30, the next event to be processed is the pending message at time 32 (message E). It is possible for the LP, once its time has been advanced to 32, to calculate that the outgoing message (associated with customer E) will have a time stamp of 70. This estimate of the outgoing time could change if the queueing discipline was not FCFS or the server could be preempted.

2.2. Characteristic Times for an Event Passing through the LP

All the definitions of lookahead (given below) can be made in terms of three characteristic times of the LP and its customers. These times are defined for each outgoing message $\Lambda$. These times are discussed below.

$t_{\text{cause}}$: The time associated with the LP when the decision to create an outgoing message is first made. (Defined by Fujimoto (Fujimoto 1988 ICPP).) Between $t_{\text{cause}}$ and the actual sending of the message $\Lambda$, its departure time (or even its departure) may be altered. Note, this definition is ambiguous if an outgoing message is caused by more than one incoming message. In such cases $t_{\text{cause}}$ is viewed as the last of these cause messages. $t_{\text{commit}}$: This is the earliest time after which NO arriving message on any port can prevent or alter the departure of the message in question, $\Lambda$. Note, $t_{\text{commit}}$ is the time at which the LP can predict all actions that the physical portal can take. $t_{\text{effect}}$: This is the time stamp on the message which is to be sent (A).

Figure 3 shows these times for a typical message, $\Lambda$. For the case of Message E, in the example above, $t_{\text{commit}}=32$ and $t_{\text{effect}}=70$.

2.3. Lookahead

It has been acknowledged that “lookahead” is an important parameter in the performance of a conservative distributed simulation since the conservative algorithm was first proposed (Chandy and Misra 1979). Unfortunately there have been a number of (slight) variations on its meaning. In all cases it can be viewed as related to the difference between the commit time $t_{\text{commit}}$ and the actual time of the event $t_{\text{effect}}$.

In the original proposal, Chandy and Misra described it in terms of $t_{\text{effect}}$: “In this case, messages (if any) sent by process $i$ to process $j$ in the interval $(t_{\text{commit}}^{i}, t_{\text{effect}}^{i})$ are independent of messages received by process $i$ after $(t_{\text{commit}}^{i})$. We [Chandy and Misra] define lookahead for the arc $(i, j)$ as $t_{\text{effect}}^{i} - t_{\text{commit}}^{i}$. The value of lookahead depends only upon $t_{\text{commit}}^{i}$ and the message histories obtained by process $i$ at $t_{\text{commit}}^{i}$.” (Chandy and Misra 1979)

It is worth noting that lookahead may be different for each outgoing arc of a process.

Prior to 1990, Fujimoto (Fujimoto 1987, Fujimoto 1988 SCS, Fujimoto 1988 ICPP) defined lookahead as: “... if a process has received all messages with time stamp $t_{\text{commit}}^{i}$ or less, and can predict all future messages with time stamp $t_{\text{effect}}^{i}$ or less, then we [Fujimoto] say the lookahead of the process is $t_{\text{effect}}^{i} - t_{\text{commit}}^{i}$ (actually, this is a simplification since different links can have different lookahead values).” (Fujimoto 1988 SCS)

He adds the proviso: “Here we [Fujimoto] assume the lookahead ability of a process is fixed throughout the simulation and is the same on each output link.” (Fujimoto 1987)

This definition restricts lookahead to the value as seen by an LP rather than as seen by a given outgoing arc. In addition, it assumes that lookahead is a fixed quantity throughout the simulation. Also these assumptions force all arcs to be treated identically.

In more recent work, Fujimoto has extended the definition of lookahead to the time varying case where lookahead is calculated for each outgoing message (Fujimoto 1990).

Wagner and Lazowska extended the definition of lookahead to include real (wall-clock) time (Wagner and Lazowska 1989). They defined lookahead as: “... if at real time $t$ an LP’s simulation clock time is $t_{\text{commit}}^{i}$, then the lookahead, denoted by $t_{\text{commit}}^{i}$, is the largest value such that at real time $t$ the LP can predict all actions that the physical portal can take. Based on all of these definitions, we feel that lookahead can be defined as $t_{\text{effect}}^{i} - t_{\text{commit}}^{i}$, given that $t_{\text{commit}}^{i}$ is taken as the earliest possible time.
2.4. Lookahead Ratio

It was decided fairly early that the absolute value of lookahead, although important for deadlock prevention (Chandy and Misra 1979), was not as critical to distributed simulation performance as its value compared to the mean time stamp increment. This lead to the quantity called “lookahead ratio” (Fujimoto 1987, Fujimoto 1988 SCS, Fujimoto 1988 ICPP, Fujimoto 1990, Wagner and Lazowska 1989). Unfortunately, again, there is some inconsistency in definition of the lookahead ratio (LAR). In general terms it is to be the ratio of the “time stamp increment” to the lookahead in a given system. The various versions are summarized below (again translated to the characteristic times defined above):

Wagner and Lazowska define lookahead ratio as a ratio of expected values (Wagner and Lazowska 1989):

\[
\text{LAR} = \frac{E\{\text{effect} - \text{cause}\}}{E\{\text{lookahead}\}}
\]

In early work Fujimoto defines LAR based on the distribution of the service times of the LP (Fujimoto 1987, Fujimoto 1988 SCS).

\[
\text{LAR} = \frac{\text{mean time stamp increase}}{\text{lookahead}}
\]

In this case the mean time stamp increase is taken to be the change in the customer’s time stamp from the time it enters the LP until it departs the LP, Lookahead (Fujimoto 1987, Fujimoto 1988 SCS) is taken as a fixed value of the minimum time stamp increment for customers passing through the LP.

In more recent work, involving the evaluation of time warp systems as well as conservative systems, Fujimoto (Fujimoto 1988 ICPP, Fujimoto 1990) has based LAR on a message by message calculation as shown below.

\[
\text{LAR} = \frac{E\{\text{effect} - \text{cause}\}}{E\{\text{lookahead}\}}
\]

We have chosen to define LAR as

\[
\text{LAR} = \frac{E\{\text{effect} - \text{cause}\}}{E\{\text{commit}\}}
\]

Note that because 0% lookahead ≤ time stamp increment, LAR is in the interval 1 ≤ LAR ≤ ∞. In order to facilitate plotting of data and to eliminate the problem with infinity, the reciprocal of LAR (inverse lookahead ratio or ILAR) is used (Preiss and Loucks 1989, Fujimoto 1990).

\[
\text{ILAR} = 1/\text{LAR}
\]

2.5. Limitations of Lookahead and Lookahead Ratio

Lookahead (any of the definitions) is based on customer bearing messages only. (i.e., events that would occur during the use of the real system.) This ignores any efficiencies (or inefficiencies) caused by null messages in a conservative synchronization technique.

A second class of limitation is caused by the definition of the time stamp increment (as used in LAR and ILAR). In this definition, there is no convenient way to handle messages which are consumed. (i.e., a \( t_{cause} \) but no direct \( t_{effect} \).) An even worse problem is what definition to use in the event that two (or more messages) are created at different times as the result of the same cause.

The definitions of LAR and ILAR are most usable in cases that customers are passed around systems (e.g., closed queueing systems). In these cases you can easily evaluate the “time stamp increment” on a customer passing through a given logical process. What is unclear is how to extend the ideas of LAR and ILAR in environments where there is no conservation of customers. Such systems can be summarized as:

Customer Consumers — LPs which consume more customers than they produce;

Customer Creators — LPs which create more customers than they consume;

and

Customer Neutral — LPs which neither create nor consume customers.

It is clear that over the duration of a given simulation a given LP may change from customer neutral to consumer or creator. One example of such a system is a two-input AND gate in a logic simulation. If one of the inputs is false for a long time, the AND gate consumes any edges passed to the gate (as messages) on its other input. Hence, it would be a customer consumer. On the other hand if one input is true for a long time, the AND gate is customer neutral with respect to edges passed to the gate on its other input.

2.6. Null Message Lookahead

One way to address the first limitation discussed above, is to consider a second type of lookahead. In a system being simulated using a conservative distributed simulation synchronization technique not only are there customer messages in the system but also null messages. In effect, a null message, with a time stamp of \( \Delta \), is a promise that the source LP will send no message with a time stamp less than or equal to \( \Delta \). (Note if the LP "knows" that no matter when the next incoming message arrives the earliest that the next outgoing message can depart is \( T \), then the LP is free to send a null message with a time stamp of \( T - \epsilon \), where \( \epsilon \) is the smallest resolvable time unit in the simulator.) Since in a conservative distributed simulation the time stamps of consecutive messages on a given arc increase monotonically, it may be argued that the further into the future a promise can be made, the sooner that simulation will be complete. Thus we define a dual to lookahead based on null messages, called null message lookahead (NML).

NML is a measure of the distance into the future that a logical process can predict all simulation messages that will be sent along an arc. NML can be specified in terms of the three characteristic times which were defined in Section 2.2. We introduce subscripts on these times to indicate whether they are times for null messages or customer bearing messages. Thus we define NML as:

\[
\text{NML} = t_{effect\ NULL} - t_{commit\ NULL}
\]

It is convenient to describe this measure in relationship to the example introduced in Section 2.1. The lookahead for customer \( E \) (which arrived at time 32 and departed at time 70) is 38 (i.e., \( 70 - 32 \)). If the writer of the model knew that there was a minimum service time of 2 then a null message at time 72 - \( \epsilon \) could be sent even if no further incoming customers were received. Thus the NML would be 40 (i.e., \( 72 - 32 \)).

2.7. Null Message ILAR

For the same reasons as discussed in Section 2.4 the raw value of NML is only part of the measure of effectiveness. In this paper we propose a second measure of simulation effectiveness, a null message inverse lookahead ratio (NILAR) based on ILAR as discussed above.

Thus we define NILAR as below.

\[
\text{NILAR} = \frac{E\{\text{NML}\}}{E\{\text{time stamp increment}\}}
\]

Which in terms of the defined characteristic times is:

\[
\text{NILAR} = \frac{t_{effect\ NULL} - t_{commit\ NULL}}{t_{effect\ CUSTOMER} - t_{cause}}
\]

The times used to compute NILAR are shown in Figure 4. Unlike ILAR which is bounded by 0 and 1, NILAR can take on any positive value. In the
case shown in Figure 4, as a side-effect of sending the customer at $t_{\text{effect CUSTOMER}}$, a null message at time $t_{\text{effect NULL}}$ can be sent. In all of the experiments described in Section 3, $t_{\text{commit CUSTOMER}}$ and $t_{\text{commit NULL}}$ are the same and referred to as $t_{\text{commit}}$.

### 3. QUEUEING NETWORK BENCHMARKS

The simulation benchmarks used to evaluate the impact of lookahead on the performance of distributed simulation are described in this section. The suite of benchmark programs used in this study was adapted from Nicol (Nicol 1988). The suite consists of simulations of four different closed stochastic queueing networks.

Each queueing network contains a fixed number of nodes (64 in all cases). A node consists of a single queue and server. Each node has a number of inputs and outputs that is determined by the topology of the network. The network is populated with a fixed number of customers. A customer arrives at a node, waits for service, receives service, and then departs for another node.

The queueing discipline at each node is FCFS. The service time at each node is a biased, exponentially-distributed random variable with a minimum service time of two ($\mu_{\text{min}}=2$) and a mean service time of ten ($\mu=10$). I.e., the service time is the sum of $\mu_{\text{min}}$ and an exponentially distributed random variable with a mean value of $\mu-\mu_{\text{min}}$. Service is non-preemptive. I.e., once a customer has commenced receiving service, the customer will complete its service at the scheduled time — no other customer can prevent the original customer from finishing on time.

The routing of customers between nodes is random. When a customer departs from a node, it selects a random output arc. Departures are uniformly distributed over all output arcs and are independent of the arc on which the customer arrived. Hence, customers carry no routing information and they are completely indistinguishable.

#### 3.1. Topologies

The four simulation benchmarks are differentiated by the topology of the interconnection between the nodes. The topologies used in this study are ring, multistage, mesh, and hypercube. Each topology is characterized by the number of input arcs per LP (fanin) and the number of output arcs per LP (fanout). (See Table II.) In all cases, the fanin is equal to the fanout.

- The ring topology contains 64 nodes. The ring is bidirectional. I.e., customers may travel clockwise or counter-clockwise.
- The multistage topology consists of eight columns of eight processors. Each node in one column is connected to two nodes in the next column. Columns are unidirectional. (The nodes of the last column are the same as those in the first.) The topology of the interconnection between columns is a perfect, two-way shuffle.
- The mesh topology is a two-dimensional, 8x8 grid of nodes. Each node is connected to four nearest neighbours. The connections between nodes are all bidirectional. The side and top edges of the grid are wrapped around to form a torus.
- The hypercube topology is a six-dimensional Boolean cube. The connections between nodes in each dimension are bidirectional.

<table>
<thead>
<tr>
<th>Network Topologies</th>
<th>Table II</th>
</tr>
</thead>
<tbody>
<tr>
<td>fanout/ topology</td>
<td></td>
</tr>
<tr>
<td>ring</td>
<td>2</td>
</tr>
<tr>
<td>multistage</td>
<td>2</td>
</tr>
<tr>
<td>mesh</td>
<td>4</td>
</tr>
<tr>
<td>hypercube</td>
<td>6</td>
</tr>
</tbody>
</table>

#### 3.2. Communication and Computation Loads

The simulation benchmarks used in this study have the characteristic that the communication and computation loads are uniformly distributed. Since the customers are indistinguishable, since routing is random, and since the service time distributions are the same in all nodes, each node will service the same number of customers during the simulation, on average. Thus, by varying the number of customers in the simulation, it is possible to vary the communication and computation load of the simulation.

The benchmark simulations in this study were done with three different sets of initial conditions. Each set of initial conditions is similar in that the same number of customers is enqueued in each node at the start of the simulation. In this paper, we present results for loads of one, four, and eight customers in each queue at time zero. This corresponds to a total of 64, 256, and 512 (respectively), customers in the system.

#### 3.3. Lookahead Implementations

In this study, we compare the performance of the benchmark systems described above using four different implementations of the queue-and-server node.

#### 3.3.1. Epsilon Lookahead

The smallest possible lookahead is zero. However, distributed simulation using conservative synchronization will immediately deadlock if all processes have zero lookahead. To avoid this deadlock, the epsilon lookahead model uses a lookahead equal to the smallest possible non-zero simulation time increment, $\epsilon$. (In Yaddes, $\epsilon = 1$.) E.g., if a customer is to depart from an LP at simulation time $t_{\text{effect CUSTOMER}}$, a message is sent from that LP to its successor with a time stamp $t_{\text{effect CUSTOMER}} - \epsilon$. When the LP’s simulation time has reached $t_{\text{commit}}$, the message can be sent.
The epsilon lookahead model does not incorporate any knowledge about the queuing discipline (FCFS) or the non-preemptive service discipline. However, in the benchmarks used in this study, it is possible to predict the precise time when a customer will depart at the instant when the customer arrives at a node. In the implementation of the epsilon lookahead model, the time of departure, \( t_{\text{effect CUSTOMER}} \), is predicted when a customer arrives at a node. However, the transmission of the customer-bearing message is delayed until the LP’s local time has advanced to \( t_{\text{commit}} \). To trigger a delayed event such as this, an LP merely sends a message to itself with time stamp \( t_{\text{commit}} \).

In the epsilon lookahead model, for every customer-bearing message sent from an LP to one of its successors, a null message with the same time stamp is sent to all the other successors of that LP. Thus, \( t_{\text{effect CUSTOMER}} \) = \( t_{\text{effect NULL}} \).

### 3.3.2. Service-Time Lookahead

The service-time lookahead model takes advantage of the non-preemptive service. Once a customer has commenced service, the departure time for that customer is fixed. As in the epsilon-lookahead model, at \( t_{\text{cause}} \) and the service-time model predicts the departure time \( t_{\text{effect CUSTOMER}} \), which is sent to all the successors of that LP. Thus, \( t_{\text{commit}} \) is the earliest the next customer can depart at time \( t_{\text{cause}} \). If a customer departs from a node at time \( t_{\text{cause}} \), then the earliest the next customer can depart is at time \( t_{\text{cause}} + \mu_{\text{min}} \), where \( \mu_{\text{min}} \) is the minimum service time. In the service-time lookahead model, a customer-bearing message will be sent from an LP at time \( t_{\text{commit}} \) with time stamp \( t_{\text{cause}} \) to one successor of that LP. At the same simulation time, null messages (non-customer-bearing) will be sent with time stamps \( t_{\text{effect CUSTOMER}} \) + \( \mu_{\text{min}} \) to all the other successors of that LP. (I.e.,

\[
\begin{align*}
& t_{\text{effect CUSTOMER}} = t_{\text{effect NULL}} + \mu_{\text{min}} \\
& t_{\text{commit}} = t_{\text{cause}} + \mu_{\text{min}} - \epsilon
\end{align*}
\]

### 3.3.3. System-Time Lookahead

This lookahead model takes advantage of the queuing discipline (FCFS) and of non-preemptive service. It is also the simplest and most natural model to implement in Yaddes. As in the previous two cases, when a customer arrives at a node, its departure time is determined. E.g., if a customer arrives at a node at time \( t_{\text{cause}} \), it will depart from the node at time \( t_{\text{cause}} + \mu_{\text{min}} = \text{service} \) where \( \mu_{\text{min}} \) is the minimum service time. In the system-time lookahead model, a customer-bearing message is sent from the LP at time \( t_{\text{commit}} \) with time stamp \( t_{\text{cause}} \) to one successor of that LP. At the same time, null messages (non-customer-bearing) will be sent with time stamps \( t_{\text{effect CUSTOMER}} \) + \( \mu_{\text{min}} \) to all the other successors of that LP. (I.e.,

\[
\begin{align*}
& t_{\text{effect CUSTOMER}} = t_{\text{effect NULL}} + \mu_{\text{min}} \\
& t_{\text{commit}} = t_{\text{cause}} + \mu_{\text{min}} - \epsilon
\end{align*}
\]

### 3.3.4. System-Time Lookahead with Presampling

The presampling lookahead model takes advantage of the queuing discipline (FCFS), the non-preemptive service, the random routing, and the inability to distinguish between customers. This model presamples the random number generator to “predict” service times and routing for customers that have not yet arrived. This model is implemented using a future list (Nicol 1988).

The presampling lookahead model treats customer-bearing messages exactly the same as the system-time lookahead model. The presampling of the random number generator to predict service times and routing only affects the time stamps on the null messages. In the presampling model, when a customer-bearing message is sent from an LP at time \( t_{\text{commit}} \) with time stamp \( t_{\text{effect CUSTOMER}} \) to a particular successor, null messages bearing time stamps \( t_{\text{effect CUSTOMER}} + i \epsilon \) are sent to all the successors, \( i \), of that LP. The prediction, \( t_{\text{prediction}} \), is obtained by computing the earliest possible time at which a customer could arrive at the \( i \)-th successor of an LP assuming an infinite number of customers arrived at that LP at time \( t_{\text{cause}} + \epsilon \). Note that the prediction is different for each output arc of the LP.

It is possible to commit to this prediction at time \( t_{\text{commit}} \) because queueing is FCFS, service is non-preemptive, routing is random, and customers are indistinguishable.

To minimize communication overhead, the prediction-bearing null messages are only sent if the prediction at time \( t_{\text{cause}} \) differs from a prediction made at an earlier time. Also note that each customer-bearing message is followed by a null message predicting the earliest possible arrival of the next customer.

### 3.4. Inverse Lookahead Ratios

Table III summarizes the formulae and values used for ILAR and NILAR for the various experiments described above. There are a number of items of note in this table.

**ILAR:** In the ILAR section notice that the impact of presampling is not apparent in the ILAR value. In fact, ILAR has saturated at 1.0 independent of load and topology. As will be apparent in Section 4, the performance of these various systems (with identical ILAR) is not identical.

**NILAR:** In the NILAR entries, as the model becomes more predictive there is a corresponding increase in NILAR. Of particular note is that with presampling, the value of NILAR is related to the fanin—fanout of the model. For each output arc, we have taken the expected NML to be the expected system time of the customer at the end of the queue \( N\mu \) plus the expected completion time for the next customer to depart on that arc. This results in the following expression for NILAR.

\[
\text{NILAR} = \frac{N\mu + \sum_{i=1}^{\infty} \left(1 - \frac{1}{f_{\text{out}}}\right)^{i-1} \left(\frac{1}{f_{\text{out}}}\right)^i}{N\mu} = \frac{N + f_{\text{out}}}{N}
\]

where:

\[
\frac{1 - \frac{1}{f_{\text{out}}}}{f_{\text{out}}}
\]

is the probability that the \( i \)-th customer is the first customer to depart on the arc.
Table III
Values of ILAR and NILAR Used
For the Analysis.

<table>
<thead>
<tr>
<th>Lookahead Technique</th>
<th>Departure Service Time</th>
<th>System Time</th>
<th>System + Pre-Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Values</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N = 1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>N = 4</td>
<td>0.25</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>N = 8</td>
<td>0.125</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NILAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Values</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N = 1</td>
<td>1.2</td>
<td>1.2</td>
<td>3.0</td>
</tr>
<tr>
<td>N = 4</td>
<td>0.105</td>
<td>1.5</td>
<td>2.0</td>
</tr>
<tr>
<td>N = 8</td>
<td>0.15</td>
<td>1.25</td>
<td>1.5</td>
</tr>
</tbody>
</table>

4. OBSERVATIONS

The simulation benchmarks described above have been implemented using the Yaddes language (Preiss et al. 1988, Preiss 1989). The simulations were run first on a uniprocessor (a single T414 Transputer) using a conventional (sequential, event-list driven) discrete event simulation kernel. The same simulations were then run on a multiprocessor consisting of eight T414 Transputers connected in a cube. Eight LPs were statically assigned to each processor. In both cases, the total execution time of the simulation was measured and the speedup (uniprocessor execution time divided by multiprocessor execution time) was calculated. Figures 5 and 6 are plots of speedup vs. ILAR and NILAR (respectively) for the various topologies, loads, and lookahead implementations.

4.1. On the Relationship between Speedup and ILAR

Figure 5 shows that under low load and with small ILAR, fractional speedup is achieved. I.e., the distributed simulation on eight processors runs slower than the sequential version of the same simulation. However, under higher loads and with larger ILAR values, very good speedup can be achieved (e.g., a speedup of 7.8 on eight processors).

In those cases where the speedup of the distributed simulation is the same as the speedup of the sequential simulation, the distributed simulation is not necessarily better than the sequential simulation. However, in those cases where the speedup of the distributed simulation is better than the speedup of the sequential simulation, the distributed simulation is generally better than the sequential simulation.

Table III shows that service-time lookahead (with load=1), system-time lookahead, and system-time lookahead with presampling all have the same ILAR (i.e., one). However, Figure 5 shows that the speedup can be substantially different even though ILAR is the same. Furthermore, the differences are not apparently correlated with fanout or load.

4.2. On the Relationship between Speedup and NILAR

Figure 6 shows that for all topologies, the best speedup is achieved for NILAR values greater than one. However, NILAR values greater than one do not necessarily imply good speedup.

Whereas ILAR does not differentiate between service-time lookahead (with load=1), system-time lookahead, and system-time lookahead with presampling, Table III shows that each of these has a different value of ILAR. Figure 6 shows that larger values of NILAR are associated with larger speedup.

In summary, the experimental results indicate that in order to achieve good speedups, the load must be high, the fanout must be small, ILAR must approach one, and NILAR must exceed one.

5. SUMMARY

It has long been recognized that the ability of a logical process to look ahead is crucial to the performance of distributed simulation using conservative synchronization. This paper presents a survey of the various definitions for look-ahead and look-ahead ratio in the literature. Unfortunately, the various definitions are not entirely consistent.

The main contributions of this paper are more precise formulations for lookahead and inverse lookahead ratio (ILAR) for customer-neutral systems and the introduction of null message lookahead and the null message inverse lookahead ratio (NILAR) as metrics for evaluating the potential of a distributed simulation implementation to achieve good speedup.

In order to assess the impact of the time stamps on null messages we introduce the concept of null message lookahead. Null message lookahead measures for null messages the same quantity that lookahead measures for customer messages. By normalizing lookahead and null message lookahead to the same quantity (i.e., the mean time stamp increment), the ILAR and NILAR ratios are directly comparable.

The empirical results presented in this paper seem to indicate that conservatively synchronized distributed simulation implementations must have ILAR values close to one and NILAR values greater than one in order to achieve good speedup. However, ILAR values close to one and NILAR values greater than one do not necessarily result in good speedups. Other requirements are higher loads and limited fanout.

6. 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 and OGP36635. We thank Ian MacIntyre for his diligence in shaking the final bugs out of the simulations described above. We also thank Jim Field for his database and graph plotting package.

7. REFERENCES


Figure 5. Speedup vs. Inverse Lookahead Ratio (ILAR).
Figure 6. Speedup vs. Null Message ILAR (NILAR).
(The legend is the same as Figure 5)