In International Journal of Computer Simulation, 1997. Journal ceased publication before article went to print.

A Framework for Estimating the Performance of Distributed Simulations

Murali S. Shanker
Kent State University

B. Eddy Patuwo
Kent State University

March 17, 1997

Authors' addresses: Murali S. Shanker, Department of Administrative Sciences, Kent State University, Kent, OH 44242, *mshankerscorpio.kent.edu*; B. Eddy Patuwo, Department of Administrative Sciences, Kent State University, Kent, OH 44242, *epatuwobsa3.kent.edu*.

This work is an extension of [29], which we presented at the *7th Workshop on Parallel and Distributed Simulation*, The Society for Computer Simulation, May 1993.

Abstract

An important aspect of running a distributed simulation is to reduce the run time of the simulation. Here we develop a framework to predict the run-time performance of conservative simulations of feed-forward networks before the simulation is run. This is done by first developing analytical models to represent the effect of the input and output waiting rules in conservative simulation on the throughput rate of messages in the simulation. A unified framework is then developed to predict run-time performance. Experiments on an Intel Hypercube suggest that the framework is quite accurate in predicting performance. By developing the framework, it is possible to evaluate the cost effectiveness of a distributed simulation without incurring the cost of the simulation.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization:**] Performance of Systems — Measurement techniques, modeling techniques; D.4.1 [**Operating Systems:**] Process Management — Scheduling; D.4.8 [**Operating Systems:**] Performance — Measurements; I.6.0 [**Simulation and Modeling:**] General; I.6.8 [**Simulation and Modeling:**] Types of Simulation — Distributed, Parallel

General Terms: Algorithms, Measurement, Performance

Additional Key Words and Phrases: Conservative simulation, approximations, performance analysis, queueing models

In recent years, distributed simulation has become a viable alternative to traditional sequential event-list simulation. As simulations of physical systems, including many analytically intractable feed-forward systems, can be time consuming, one potential use of parallel simulation is to reduce run time. For example, in manufacturing it is essential to understand the impact of different system parameters like the number of parallel servers, number of inspection stations, the allocation of tasks to workstations, the effect of buffer space and the assignment of workers on the throughput of jobs in the system so that efficient flow lines can be designed. For such problems, as simulation is often the only practical tool for analysis, distributed simulations can be used to reduce run time. But partly because of the expertise needed to run distributed simulations (DS) efficiently, it has not yet enjoyed widespread use [11, 31]. This work develops a framework for predicting distributed simulation run-time performance before the simulation is run. The framework can then be used as a tool to evaluate the cost effectiveness of running a distributed simulation, and to develop performance-enhancing strategies, e.g., adaptive schemes, in a manner transparent to the end user.

Two popular approaches for distributed simulation (DS) are the conservative [4, 5, 6, 22] and the optimistic paradigms [13]. Numerous studies have been conducted to study the performance of the above two basic methods (see [11] for a summary of selected speedup measurements). To a large extent, the performance of the distributed simulation method is affected by the overheads necessary to ensure a correct simulation. For conservative simulations, the overheads are the *input* and *output* waiting rules that are needed to ensure proper timestamp ordering of messages [10, 17, 19, 32], while for optimistic simulations it is the cost of rollbacks and state saving [9, 10]. As such, various methods have been proposed for reducing overheads in distributed simulations [3, 8, 10, 12, 18, 20, 23, 24, 25, 33]. But, the trade-off between seeking an optimized method and reducing overheads is still not clear.

Our work parallels Lin and Lazowska's work in conservative simulations [19], where they study the effect of input and output waiting rules on simulation performance. As in [19], we use the Chandy-Misra-Bryant (CMB) [4, 5, 6, 22] algorithm, but here we develop analytical models for *predicting* the effect of input and output waiting rules on the throughput rate of messages in conservative distributed simulations. The models are then incorporated in a framework that can be used to estimate run-time performance for conservative simulations of feed-forward network systems *before* the simulation is run. In so doing, we may be able to improve the simulation's run time performance by identifying potential bottlenecks in the system before the system is run.

The rest of the paper is organized as follows: The next section briefly reviews the CMB algorithm, and identifies the objectives of our research. Section 2 describes the overheads in a conservative

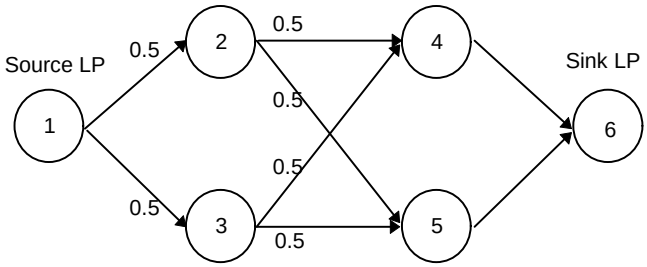


Figure 1: A logical system with 6 LPs

simulation, and presents mathematical models to determine their effect on the performance of DS. Section 3 provides a framework for predicting the performance of DS. Experimental analysis is in Section 4, while Sections 5 and 6 contain the results and conclusions, respectively. In Appendix A we discuss how lookahead can be used more effectively to improve simulation performance, while Appendix B defines the variables used in this paper.

1 The Chandy-Misra-Bryant Algorithm

In this model, the physical system to be simulated is represented as a collection of physical processes (PPs) that interact through messages. To simulate the system, each PP is represented by a logical process (LP) in the logical (simulated) system. The logical system correctly simulates the physical system if it predicts the exact sequence of message transmissions in the physical system. That is, if $ts_1, ts_2, \cdots$ are the times at which the messages $m_1, m_2, \cdots$ are transmitted in the physical system and $ts_1 \leq ts_2 \leq \cdots$, then the logical system should be able to output the message sequence $\{(ts_1, m_1), (ts_2, m_2), \cdots\}$.

In DS, there is no global event list or a global simulation clock. Here each LP has its own clock indicating how long the LP has been executing in *simulation time*, and events in a sequential simulation are replaced by timestamped messages that go from LP to LP along directed channels in the logical system. As LPs in the logical system may be assigned to different processors in the computer system, to correctly simulate a PP, the corresponding LP must process messages in timestamped order, as opposed to real-time arrival order. As such, a major difficulty in parallel simulation is to ensure that event causality is maintained. In conservative simulation this is accomplished by ensuring that an LP does not process a message until it is certain that it will not receive an earlier (timestamped) message.

Consider an example of a CMB simulation for a feed-forward network. Messages originate at LPs

called *source* LPs. They are then sent to the next LP in the logical structure, where they are processed and then sent to the next LP, etc., until finally they arrive at a *sink* LP and leave the system. For example, Figure 1 shows a logical system with six LPs. There LP_1 is the source LP, and LP_6 the sink LP. At each LP, there are three queues: an *input* queue for input messages, a *local event* queue for events scheduled for the LP itself, and an *output* queue for output messages (in practice, the local event and output queues are combined). Messages in the above queues are sorted in non-decreasing timestamp order. Initially, some events are scheduled in the local event queues of LPs (source LPs must have non-empty event queues initially). Then each LP, say LP_N, repeats the following steps:

Step 1: LP_N waits until at least one message from each input channel is in its input message queue. Let m_k, with timestamp ts_k, be the message with the smallest timestamp among all waiting messages (for source LPs, there are usually no input queues, then m_k is the message with the smallest timestamp in the local event queue). Thus LP_4 in Figure 1 would wait until it received messages from both LP_2 and LP_3.

Step 2: LP_N processes, in order, all events in its local event queue with timestamp no greater than ts_k, as well as the message m_k itself. The processing of these messages may generate new messages and events, which should in turn be processed if their timestamps are no greater than ts_k.

Step 3: LP_N then processes, in order, all output messages with timestamp no greater than $ts_k + d$, where d is the lookahead. For sink LPs, there are no output queues, and hence no output message is sent.

Step 4: If m_k was an *end-of-simulation* (EOS) event, then LP_N terminates by sending an EOS message with timestamp ∞ along each output channel. Otherwise, return to Step 1.

For such simulations, in addition to the execution time incurred by a message at an LP (we say that a message is executed at an LP to mean that it is really executed on the physical processor to which the LP is assigned), messages may also be delayed for the following reasons:

- In Step 1 of the above algorithm, messages at LP_N are blocked from further processing until messages exist in all input queues to LP_N. This waiting is called the *input waiting rule* (IWR), and we say that the waiting messages have not yet satisfied their *precedence* relationships. Note that for LPs with a single input queue, precedence relationships are satisfied immediately.

- Messages that have satisfied their precedence relationships wait in a *ready* queue until the processor becomes free to execute them. In practice, the above two waiting periods are usually combined. Here, the *ready* queue is used to model and differentiate between the two types of waiting in the input queue.

- In Step 3, messages wait in an *output* queue until the LP's clock value advances to the message's timestamp. This is needed to ensure that messages are sent in non-decreasing timestamp order. This waiting is known as the *output waiting rule* (OWR).

As the execution cost of a message at an LP is normally a fixed cost (assuming homogeneous processors and ignoring communication costs), an optimal implementation of the algorithm would seek to minimize the input and output waiting overheads [16]. Here we provide a framework to predict the run-time performance of such conservative distributed simulations before the simulation is run. By run-time performance we mean the message *throughput rate*, i.e., the long-run time average rate at which messages leave the system. If δ denotes this rate, then $\delta = \lim_{t\to\infty}$(number of messages that leave the system in the first t time units). As throughput rate is inversely related to run time, two implementations of a logical system can be compared by observing their throughput rates. We predict throughput rates, rather than the actual run time of a simulation, as it is a more tangible objective for the purposes of implementing performance-enhancing tools [27]. Therefore, the objective of our framework is "to predict the throughput rate of messages for conservative simulations of systems with feed-forward network topology before the simulation is run."

The next section defines the waiting overheads in a conservative simulation and develops models to predict their effect on the throughput rate of messages.

2 Waiting Overheads in Conservative Simulations

The waiting overheads in most conservative simulations arise primarily because of the input and output waiting rules (Section 1). We formally define the waiting overheads as follows:

Definition 1 Let there be $n \geq 1$ input channels to an LP, say LP_N. Let m_i, $1 \leq i \leq n$, be the message from input channel i that LP_N waits to receive, and tr_i be the real time that LP_N waits before receiving m_i. tr_i is equal to 0, if m_i is already in the input channel. Then the time LP_N waits before it can execute the next input message m_i is $t_{\max,m_i} = \max\{tr_i;\ 1 \leq i \leq n\}$ (Step 1 of CMB algorithm). Then, $t_{\max,m_i}$ is the input waiting overhead of the CMB algorithm.

Definition 2 In the CMB algorithm, LP_N sends an output message m_{out} with timestamp ts_{out} when LP_N's clock value equals ts_{out} (Step 3 of CMB algorithm). Let tr_{out} be the real time that elapses between m_{out} joining LP_N's output queue and until LP_N's clock value advances to ts_{out}. Then tr_{out} is the output waiting overhead of the CMB algorithm for message m_{out}.

There are many instances when waiting overheads can be reduced by taking advantage of the *event knowledge* in the system [8, 10, 20, 23, 33]. For example, in first-in first-out systems the output waiting overhead can be reduced to zero, as an output message can be sent as soon as it is generated. Similarly, when lookahead is present, input messages may be processed out-of-order as long as care is taken to send output messages in the right timestamp order. Initially, in developing our framework, we do not exploit any event knowledge that may be inherent in the system. In Appendix A we relax this assumption and show how lookahead capabilities may be used more effectively to improve simulation performance.

The following convention is used for naming variables: variables with a superscript "L" denote logical time values, otherwise time units refer to physical time. Thus, if ν denotes the expected physical service time for a message, then ν^L denotes the expected logical service time for a message. In addition, we use the dot "." notation to show summation over the corresponding subscript. For example, if λ_{ij} is the arrival rate of messages to LP_j from LP_i, then $\lambda_{.j} = \sum_i \lambda_{ij}$ would be the superposition arrival rate of messages to LP_j, where the summation is over all input channels to LP_j. A complete list of variable definitions is given in Appendix B. Finally, as our interests lies in estimating throughput rates, variable values in our examples and experiments (given in the following sections) are presumed to estimate long-run time averages, i.e., as time $\rightarrow \infty$. Also, while we do not explicitly define variables to differentiate between estimators and parameters, this distinction should be clear from the problem context.

The next section presents an analytical model that predicts the effect of the input waiting rule on the throughput of messages in the system.

2.1 A Model to Predict the Effect of Input Waiting Rule on Performance

In Step 1 of the CMB algorithm, the message that is chosen from the n input channels to an LP is the one that has the least timestamp of all waiting messages. If that message has not yet arrived (i.e., the channel is empty), then messages in the other input channels are blocked from further processing. Clearly, this synchronization protocol enforces a penalty on the throughput of messages. To model

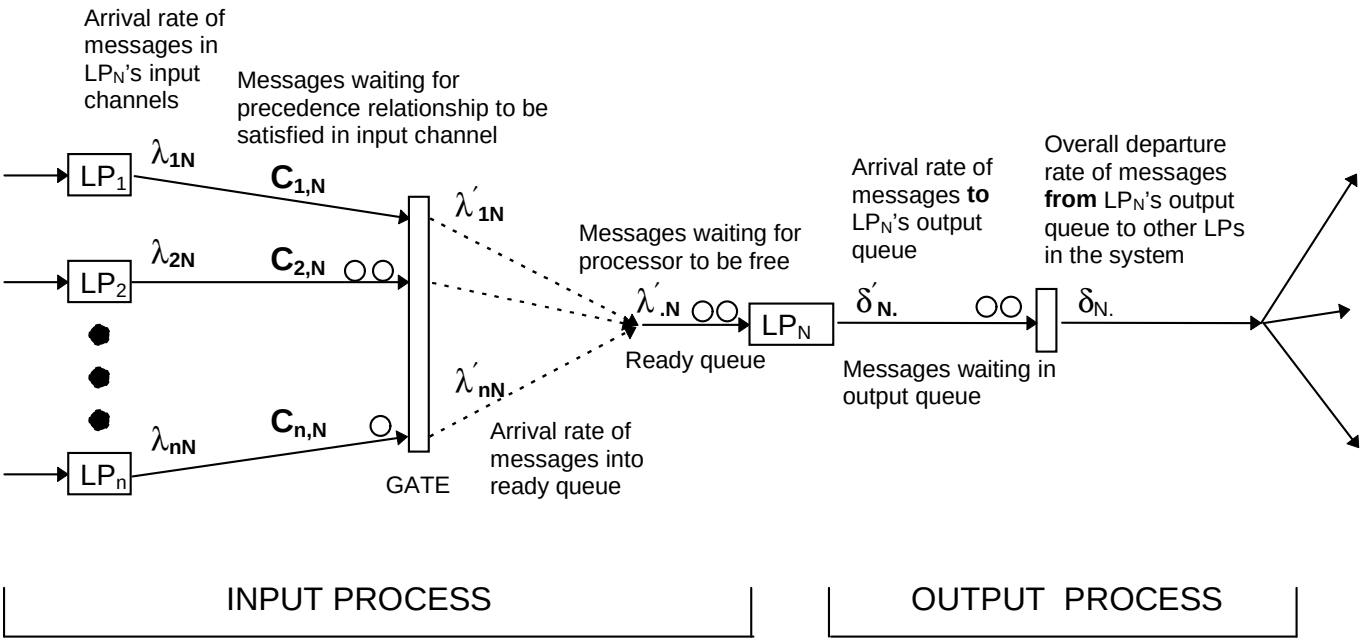


Figure 2: Input and output process at an LP in a logical system

the effect of this penalty consider Figure 2, which depicts the input processes to an LP in the logical system. Here LP_i, $i \in \{1, 2, \cdots, n\}$, sends messages to LP_N along directed input channel $\mathrm{C}_{i,N}$ at rate λ_{iN}. The n input channels, $\mathrm{C}_{i,N}$, $i \in \{1, 2, \cdots, n\}$, merge into a *gate*. There is a single queue, called the *ready* queue, connecting the gate to the node LP_N, and let λ'_{iN} denote the arrival rate of messages into the ready queue from $\mathrm{C}_{i,N}$. Messages to LP_i, $i \in \{1, 2, \cdots, n\}$, are assumed to arrive from other LPs in the system, or originate at LP_i, if LP_i is a source LP.

The gate models the IWR as follows: a message in $\mathrm{C}_{i,N}$ passes through the gate if it is the first message there (messages arrive in non-decreasing timestamp order along each channel), and if the gate is *open* to that channel. At any point, the gate is open to only one channel: the channel that has either received, or will receive the message with the least timestamp among messages in all input channels. Once the message joins the ready queue, the gate resets to the channel that will receive the message that *should* be processed next. Thus the gate acts to ensure that messages join the ready queue in non-decreasing timestamp order. The gate is an *artificial* device used to differentiate between the two different input waiting periods for a message. Messages before the gate are waiting for precedence relationships to be satisfied, while messages in the ready queue are waiting for the processor to become free. If the IWR imposes no penalty on throughput, then the arrival rate of messages into the ready queue will be the superposition rate of the n input arrival processes. That is, if $\lambda'_{.N}$ defines this

superposition arrival rate to the ready queue, then $\lambda'_{.N} = \sum_{i=1}^{n} \lambda'_{iN} = \sum_{i=1}^{n} \lambda_{iN}$. But, in conservative methods there is usually a penalty to ensure proper timestamp ordering, and as such the arrival rate to the ready queue is likely to be less than the above superposition rate. This indicates an unstable queue. Therefore, in a practical implementation of this simulation some form of flow control will be needed for the simulation to complete normally [14]. For example, mechanisms like time windows [30] and bounded lag [21] are used for queue stabilization. We ignore sure implementation overheads. Therefore, our approach of modeling the effect of the IWR on performance points to certain fundamental limits on throughput imposed by the IWR. This is reflected in our experimental results (Section 5).

While extensive work has been done in developing approximations for superposition processes for systems with general arrival streams with a general service distribution (i.e., $\sum GI/G/1$ systems), and a prespecified queue discipline like FIFO [1, 2, 34, 35, 36], to our knowledge, no results yet exist for systems with queue discipline as defined by precedence relationships like those in DS. In an earlier paper [29], mainly using extensive simulations, we developed a first-moment approximation for the superposition process in a DS. Our results have since been supported in an analysis of a formal stochastic model by Kumar and Shorey [14]. In this paper, we extend our results of [29] to develop a framework to predict simulation performance.

To present our approximation for the superposition process arising in DS, we consider an equivalent interpretation for the process of Figure 2. As mentioned earlier, let messages arrive at rate λ_{iN} along channel $\mathrm{C}_{i,N}$. Let p_{iN} ($\sum_{i=1}^{n} p_{iN} = 1$) represent the steady-state probability that the gate is *open* to $\mathrm{C}_{i,N}$. If messages are waiting in that channel, the first waiting message passes through the gate into the ready queue. If there are no messages in $\mathrm{C}_{i,N}$, the gate *remains open* to that channel until a message arrives and passes through it. The gate then with probability p_{jN}, independent of the previous state, waits for a message along $C_{j,N}$. This interpretation does away with timestamped messages, and the effect of precedence relationships is mimicked by the *probability gate*.

To provide insight into our approximation (given below), consider Figure 2 with two input channels, i.e., $n = 2$. Let successive messages arrive along channel $\mathrm{C}_{1,N}$ with timestamps $1, 2, 3, \cdots$, and in $\mathrm{C}_{2,N}$ with timestamps $2, 4, 6, \cdots$. Then, independent of the physical arrival rate, for every message that joins the ready queue from $\mathrm{C}_{2,N}$, two join from $\mathrm{C}_{1,N}$ (Step 1 of CMB algorithm). Therefore, an estimate of the steady-state probability of a message joining the ready queue from each of the input queues is $p_{1N} = 2/3$ and $p_{2N} = 1/3$. For this system to be stable in $\mathrm{C}_{2,N}$, the physical arrival rate λ_{1N} of messages in $\mathrm{C}_{1,N}$ must be at least twice λ_{2N}, the physical arrival rate of messages in $\mathrm{C}_{2,N}$. If not, $\lambda'_{2N} < \lambda_{2N}$, and $\mathrm{C}_{2,N}$ experiences an increasing queue as the simulation progresses. Conversely,

if λ_{1N} is more than twice λ_{2N}, $C_{1,N}$ experiences an increasing queue with $\lambda'_{1N} < \lambda_{1N}$. For this system to be stable, then, λ_{1N} should be exactly twice λ_{2N}. When this is so, the *timestamp increment per unit physical time* will be identical across the two input queues, and $\lambda'_{.N} = \sum_{i=1}^{n} \lambda'_{iN} = \sum_{i=1}^{n} \lambda_{iN}$. We formalize this insight in our model given below. Then, an approximation for the arrival process to a ready queue in a conservative distributed simulation is:

$$\begin{aligned} \lambda'_{iN} &= \frac{p_{iN}}{(p_{mN}/\lambda_{mN})}\,, \\ \text{where } \frac{p_{mN}}{\lambda_{mN}} &= \text{Maximum}\{\frac{p_{iN}}{\lambda_{iN}} \ : i \in \{1, \cdots, n\}\} \end{aligned} \tag{1}$$

Then,

$$\lambda'_{.N} = \sum_{i=1}^{n} \lambda'_{iN} = \sum_{i=1}^{n} \frac{p_{iN}}{(p_{mN}/\lambda_{mN})} = \frac{1}{(p_{mN}/\lambda_{mN})} \tag{2}$$

To determine the logical superposition arrival process into the ready queue, we proceed as follows: For a logical time period of T^L, the number of messages taken from each input channel is T^L/τ^L_{iN}, where τ^L_{iN} is the expected logical timestamp increment in C_{iN}. Then, an estimation of the logical arrival rate into the ready queue is

$$\begin{aligned} \lambda^L_{.N} &= \frac{\text{Number of messages taken in that time period}}{\text{Time period}} \\ &= \frac{\sum_{i=1}^{n}(T^L/\tau^L_{iN})}{T^L} \\ &= \sum_{i=1}^{n} 1/\tau^L_{iN} \\ &= \sum_{i=1}^{n} \lambda^L_{iN} \end{aligned} \tag{3}$$

Equations (1) and (2) show that the departure rate λ'_{iN} of messages from $C_{i,N}$ is *restricted* by the channel that observes p_{mN}/λ_{mN}. This channel, i.e., the channel that observes p_{mN}/λ_{mN}, represents the channel with the *least* logical timestamp increment rate (TSIR). To show this, let TSIR_{iN} denote the timestamp increment rate in input channel C_{iN}. Then

$$
\begin{aligned}
\text{TSIR}_{iN} &= \text{Expected logical time increment in } \text{C}_{iN}\text{'s clock per unit physical time} \\
&= \frac{\text{Expected logical time change}}{\text{Expected time between arrival of messages in channel } \text{C}_{iN}} \\
&= \frac{\tau^L_{iN}}{1/\lambda_{iN}} \\
&= \frac{\lambda_{iN}}{\lambda^L_{iN}} \qquad (4)
\end{aligned}
$$

We estimate p_{iN}, the probability of choosing a message from C_{iN}, as:

$$
\begin{aligned}
p_{iN} &= \frac{\text{Logical arrival rate of messages in } \text{C}_{iN}}{\text{Sum of logical arrival rate of messages across all input channels}} \\
&= \frac{\lambda^L_{iN}}{\sum_{i=1}^{n} \lambda^L_{iN}} \qquad (5)
\end{aligned}
$$

Then, substituting Equations (3) and (5) into (4)

$$
\begin{aligned}
\text{TSIR}_{iN} &= \lambda_{iN} \times \frac{1}{p_{iN} \sum_{i=1}^{n} \lambda^L_{iN}} \\
&= \frac{1}{\lambda^{L'}_{.N}} \times \frac{\lambda_{iN}}{p_{iN}} \qquad (6)
\end{aligned}
$$

From Equation (6), we see that p_{mN}/λ_{mN} (given in Equation (1)) refers to $\min\{\text{TSIR}_{iN} : i = 1, \cdots, n\}$, or to the channel with the least timestamp increment per unit time.

Each time a distributed simulation is run, the objective is to reduce the run time of the simulation. As run time is directly related to the throughput of messages in the system, any approach to increase this throughput should have a corresponding decrease in run time. Equations (1) through (5) allow us to model the effect of the IWR on throughput of messages. Some important implications are:

- The throughput of messages is restricted by the channel with the least timestamp increment per unit time. That is, the channel observing p_{mN}/λ_{mN}. Therefore to improve performance, the objective is to minimize p_{mN}/λ_{mN}, or equivalently, to increase the TSIR in that channel.

- The greater the inequality in timestamp increment rates among the input channels, the higher the penalty of the IWR. The penalty is minimized when $p_{iN}/\lambda_{iN} = p_{jN}/\lambda_{jN}, \forall i, j$.

- When $p_{iN}/\lambda_{iN} \neq p_{mN}/\lambda_{mN}$, $\lambda'_{iN} < \lambda_{iN}$. That is, for channels with a timestamp increment rate (TSIR) greater than the minimum TSIR among the input channels, the departure rate of messages through the gate will be less than the arrival rate to the gate. This signifies an increasing queue possibly leading to memory exhaustion.

Clearly, to reduce the impact of the input waiting rule it is necessary to equalize the timestamp increment rate across input channels. Two parameters can be used to do this: the probability of choosing from an input channel, and the physical arrival rate of messages. The order in which messages are picked from input channels (i.e., the probability of choosing a message) is determined solely by the timestamps of the messages. For most simulations, this value cannot be changed, as usually the mean timestamp change at an LP is a parameter that is fixed for the simulation. This is a disadvantage as this parameter can no longer be manipulated to increase the efficiency of the simulation. But as shown in Section 3, this fixed value is also advantageous as it can be used to predict simulation performance *a priori*. On the other hand, the arrival rate of messages can be changed during the simulation by a suitable allocation of LPs to processors [26]. It is beyond the scope of this paper to provide allocation strategies that use the framework to improve simulation performance. Here we only show how the framework predicts performance based on the characteristics of a particular assignment of LPs to processors. Allocation strategies can then use this information to improve performance.

The above section has discussed the effect of the IWR on the throughput of messages. Models have been developed to discuss this impact. The next section develops models to determine the effect of the output waiting rule (OWR) on throughput.

2.2 Predicting the Effect of Output Waiting Rule on Performance

In Step 3 of the CMB algorithm, messages wait in an LP's output queue until their timestamp values are less than or equal to the timestamp of the next input message to be processed by that LP (ignoring any lookahead). This implies that the OWR that ensures proper timestamp ordering of output messages may impose a penalty on the rate at which messages are sent from an LP's output queue to other LPs in the system. A graphical representation of this output process is shown in Figure 2. Here messages to LP_N arrive at rate $\lambda'_{.N}$, and are serviced at rate μ_N (again, service at an LP implies that it is serviced on the processor to which the LP is assigned). If $\delta'_{N.}$ is the rate at which messages join LP_N's output queue, then clearly $\delta'_{N.} = \min\{\lambda'_{.N}, \mu_N\}$. If the output waiting rule imposes *no* penalty on throughput, then the rate at which messages leave LP_N's output queue, i.e., $\delta_{N.}$, will be equal to $\delta'_{N.}$. But in conservative simulations, this will rarely be the case.

As an example to illustrate the effect of the OWR, consider Table I, which shows a simulation of the input and output process at an LP, say LP_N, for 10 physical time periods. (Again, for the purposes of illustration, we assume that the average rates calculated from Table I estimate the long-run time average behavior of the simulation.) It is assumed that the service time at that LP is negligible, and

Table I: Departure process from an LP's output queue

Real Time	Input Msg. Arrival	Output Msg. Scheduled	LP's Clock (logical)	Output Msg. Sent	Messages in Output Queue
1	$(m_1, 10)$	$(m'_1, 35)$	10	—	$(m'_1, 35)$
2	$(m_2, 20)$	$(m'_2, 30)$	20	—	$(m'_2, 30), (m'_1, 35)$
3	$(m_3, 30)$	$(m'_3, 90)$	30	$(m'_2, 30)$	$(m'_1, 35), (m'_3, 90)$
4	$(m_4, 40)$	$(m'_4, 50)$	40	$(m'_1, 35)$	$(m'_4, 50), (m'_3, 90)$
5	$(m_5, 50)$	$(m'_5, 75)$	50	$(m'_4, 50)$	$(m'_5, 75), (m'_3, 90)$
6	$(m_6, 60)$	$(m'_6, 105)$	60	—	$(m'_5, 75), (m'_3, 90), (m'_6, 105)$
7	$(m_7, 70)$	$(m'_7, 115)$	70	—	$(m'_5, 75), (m'_3, 90), (m'_6, 105),$ $(m'_7, 115)$
8	$(m_8, 80)$	$(m'_8, 135)$	80	$(m'_5, 75)$	$(m'_3, 90), (m'_6, 105), (m'_7, 115),$ $(m'_8, 135)$
9	$(m_9, 90)$	$(m'_9, 150)$	90	$(m'_3, 90)$	$(m'_6, 105), (m'_7, 115), (m'_8, 135),$ $(m'_9, 150)$
10	$(m_{10}, 100)$	$(m'_{10}, 165)$	100	—	$(m'_6, 105), (m'_7, 115), (m'_8, 135),$ $(m'_9, 150), (m'_{10}, 165)$

(m_i, t) implies input message i at logical time t

(m'_i, t) implies output message i at logical time t.

therefore an output message is scheduled instantly on receipt of its corresponding input message. For this simulation, from Table I, the estimated physical arrival rate of messages in the input queue and the output queue are the same, i.e., $\lambda'_{.N} = \delta'_{N.} = 1$. But, the estimated expected logical timestamp between messages, calculated by observing the logical time between messages in Table I, in the input queue is $1/\lambda^{L'}_{.N} = 10$, and in the output queue is $1/\delta^{L'}_{N.} = 15$. The effect of the OWR can be noted by calculating the rate at which messages are *sent* from LP_N's output queue. Here, the estimated expected time between departures is $1/\delta_{N.} = \frac{(4-3)+(5-4)+(8-5)+(9-8)}{4} = \frac{3}{2}$. Thus, $\delta_{N.} = 2/3 < \delta'_{N.}(= 1)$. This clearly is an unstable situation, and some form of interprocessor flow control will be needed in a practical implementation for the simulation to complete normally. As we ignore such overheads here, our analysis leads to certain fundamental limits on throughputs imposed by the OWR.

Two factors affect the departure rate of messages waiting in LP_N's output queue: the physical arrival rate of messages to LP_N's input and output queues, and the relative logical timestamp of messages in the input and output queues. For example in Table I, $(m'_2, 30)$ will wait in LP_N's output queue until $(m_3, 30)$ arrives at physical time 3. If $(m'_2, 30)$ has not yet been scheduled in the output queue when $(m_3, 30)$ arrives, the departure of $(m'_2, 30)$ from LP_N is further delayed. Secondly, $(m'_2, 30)$ is sent only when its timestamp is less than or equal to the timestamp of the next input message to be processed, here $(m_3, 30)$. Therefore, messages from the input and output queues are processed in non-decreasing timestamp order. The above conditions mirror those of the IWR where messages in input channels wait until their timestamps are the least among all future message timestamps. This suggest that the approximations and results of Section 2.1 may be applicable for determining the penalty of OWR on throughput. Specifically, for the simulation of Table I consider applying Equations (1) through (5) with $n = 2$ and channel $\mathrm{C}_{1,N}$ representing the input (ready) queue and $\mathrm{C}_{2,N}$ the output queue. Then, the departure process from $\mathrm{C}_{2,N}$ through the gate would represent the departure process of messages from LP_N's output queue. That is, $\lambda'_{2,N}$ would represent $\delta_{N.}$. Using data from Table I, and Equations (1), (2) and (5): $\lambda_{1N} = \lambda_{2N} = 1$, $\lambda^L_{1N} = 1/10$, $\lambda^L_{2N} = 1/15$, $p_{1N} = \frac{1/10}{(1/10+1/15)} = \frac{3}{5}$, $p_{2N} = \frac{1/15}{(1/10+1/15)} = \frac{2}{5}$, $\frac{p_{mN}}{\lambda_{mN}} = \max\{\frac{3}{5}, \frac{2}{5}\} = \frac{3}{5}$. Then, $\delta_{N.} = \lambda'_{2N} = \frac{(2/5)}{(3/5)} = \frac{2}{3} < \lambda_{2N}(= 1)$.

Exactly the same results as obtained directly from Table I. As in Section 2.1, it can be shown that the departure rate from the output queue is *limited* by the queue with the least timestamp increment rate. Here the TSIR for the input queue is 10 (using Equation 4) and for the output queue it is 15. As the output queue TSIR is greater than the input queue TSIR, the departure process from the output queue is restricted by the input arrival process. Therefore, the departure rate *from* the output queue is less than the arrival rate *into* the output queue. As in Section 2.1, equalizing timestamp increment

rates between the input and output queues reduces the effect of the OWR. From this discussion, we state the following:

Lemma 1 Let λ_I and λ_I^L denote the physical and logical arrival rate of messages to an LP, say LP_N, respectively. Also, let λ_O and λ_O^L denote the physical and logical arrival rate of messages **to** LP_N's output queue, respectively. Then, an approximation to δ, the expected rate at which messages **leave** LP_N's output queue, is:

$$\begin{aligned} \delta &= \frac{p_O}{\max\{\frac{p_I}{\lambda_I}, \frac{p_O}{\lambda_O}\}} \\ &\text{where} \\ p_I &= \frac{\lambda_I^L}{(\lambda_I^L + \lambda_O^L)} \\ p_O &= \frac{\lambda_O^L}{(\lambda_I^L + \lambda_O^L)} \end{aligned}$$

With reference to Figure 2, λ_I and λ_I^L refer to $\lambda'_{.N}$ and $\lambda^{L'}_{.N}$, the physical and logical arrival rate into the ready queue, respectively, and λ_O and λ_O^L to $\delta'_{N.}$ and $\delta^{L'}_{N.}$, the physical and logical arrival rate into the output queue, respectively. The above lemma suggests that the approximations of Section 2.1 could be used to determine the penalty of the OWR if we know the arrival processes into the input and output queues. Knowing the arrival and service processes at an LP, the arrival process into the output queue can be determined simply as:

Lemma 2 Let $\lambda'_{.N}$ and $\lambda^{L'}_{.N}$ denote the physical and logical arrival rates to LP_N, and μ_N and μ_N^L denote the physical and logical service rates at that LP. Then, $\delta'_{N.} = \min\{\lambda'_{.N}, \mu_N\}$ and $\delta^{L'}_{N.} = \min\{\lambda^{L'}_{.N}, \mu^L_N\}$, where $\delta'_{N.}$ and $\delta^{L'}_{N.}$ refer to the physical and logical arrival rates of messages into LP_N's output queue, respectively.

In their paper on waiting overheads in conservative simulations [19], Lin and Lazowska, for specific distributions of arrival and service processes, derive exact results for the waiting time of messages in the output queue. They also show that it is possible to determine $E(N_N)$, the expected number of input messages needed at LP_N before an output message is sent. Given their results, and under the assumption that $E(N_N)$ can be determined, the effect of the OWR on throughput can alternatively be stated as follows:

Lemma 3 If $\delta'_{N.}$ is the rate at which messages arrive into LP_N's output queue, then the departure rate of messages from LP_N's output queue is $\delta_{N.} = \frac{\delta'_{N.}}{E(N_N)}$.

In the simulation of Table I, an estimate of $E(N_N)$ is 15/10 (ratio of the expected logical time between arrival of messages in the output queue and input queue). Using this estimate and the above lemma, $\delta_{N.} = \frac{\delta'_{N.}}{E(N_N)} = 1/(3/2) = 2/3$. The same result as obtained earlier. If $E(N_N)$ can be determined, Lemma 3 provides an exact answer to the throughput rate from an LP's output queue. But in practice, the assumptions of Lin and Lazowska's [19], i.e., exponential or uniform time between arrivals and exponential service time, are unlikely to be satisfied. In such cases, Lemma 1 can be used to determine the effect of the OWR on throughput.

This section has provided ways of determining the effect of the OWR on throughput. The next section uses the results of Sections 2.1 and 2.2 to develop a framework to predict simulation run-time performance.

3 A Framework for Estimating Run-Time Performance

Our framework to predict distributed simulation performance is based on the premise that given the arrival processes of messages from source LPs into the system, the arrival and departure processes at all other LPs can be predicted recursively. As the rate at which messages are deleted from the various LPs in the system determines the throughput rate for the simulation, the run-time performance of the simulation can be estimated.

The framework is shown in Figure 3, with supporting routines in Figures 4 through 6. Figure 4 shows the algorithm for estimating the input processes to an LP, and uses results from Section 2.1. The algorithm for estimating departure processes from an LP uses results from Section 2.2, and is given in Figure 5, while Figure 6 shows the estimation procedure at source LPs. The complete framework is illustrated below, and makes the following assumptions:

Assumption 1 The distributions of logical service time at each LP and of logical time between arrival of messages originating from source LPs are known. Specifically, only the first moment of the above distributions need be known. As the expected logical service time and logical time between arrival of messages that originate (are generated) at source LPs are input parameters to most simulations, this information is readily available.

Assumption 2 For each LP, the *relative* magnitudes of the physical arrival and service rates can be estimated. That is, for each LP we can determine if the physical arrival rate is greater than the service rate there, or vice versa. The actual values of the arrival and service rates need not be known. Note

Table II: Logical service rate for logical system of Figure 1

LP	LP Type	Expected Service Rate	Predecessor LPs	Successor LPs
1	Source	1	—	2, 3
2	—	$\frac{1}{4}$	1	4, 5
3	—	$\frac{1}{2}$	1	4, 5
4	—	1	2, 3	6
5	—	1	2, 3	6
6	Sink	1	4, 5	—

that the relationship between arrival and service rates is affected by the assignment of LPs to processors and the polling system used to allocate processing time among various LPs in that processor. As this is implementation specific, we only assume that it is possible to determine the relative magnitudes of arrival and service rates for each LP in that system for any particular assignment of LPs to processors.

Assumption 3 The relative magnitudes of communication and processing rates are known. Knowing this, the effect of communication on throughput can be modeled.

As an illustration, consider the logical system of Figure 1 with logical service rates as given in Table II. This logical system can be viewed as an interconnected graph with directed arcs connecting LPs. If there is a directed arc from LP_i to LP_j, then we say LP_i is a *predecessor* of LP_j, or that LP_j is a *successor* to LP_i. Table II shows the predecessor and successor LPs for each LP of Figure 1, and lists LPs in precedence order. That is, all predecessor LPs of LP_i appear before it in the list.

A step-by-step calculation of the throughput process for the logical system of Figure 1 is shown in Table III, and is discussed below. Note that *no* assumption is made about the assignment of LPs to processors. Instead we assert that the performance calculations shown in Table III are valid for *any* assignment that satisfies the following problem specific assumptions, which are made for convenience in illustrating the example:

Assumption 4 The physical arrival rate at each LP is less than or equal to the physical service rate at that LP. That is, $\lambda'_N \leq \mu_N$; $\forall N$. This relates to Assumption 2 of our framework.

Assumption 5 The rate at which messages originate from source LPs is given by the algorithm in Figure 7. Then, the physical arrival rate of messages at source LPs will be less than or equal to the

```
MAIN();

/*Main routine for estimating the throughput process in a distributed simulation*/

Step 1      Sort LPs according to precedence order. That is, all predecessor LPs to LP_j
            should appear before it in the list. By definition, source LPs have no predecessors.

For each LP_j taken in order from the list DO
    if LP_j == source LP
        call source_LP_calculation(LP_j);
        call estimate_departure_rate(LP_j);
    else
        call estimate_arrival_rate(LP_j);
        call estimate_departure_rate(LP_j);
    endif
end_DO

end_MAIN
```

Figure 3: Algorithm to estimate performance of distributed simulations

SUBROUTINE Estimate_Arrival_Process (LP_j);

/*Estimates arrival processes to an LP*/

/*Estimate arrival process along each input channel*/

$\lambda_{.j}^{L'} = 0$; /* initialize logical arrival rate for superposition process*/

For *each* input channel to LP_j from LP_i DO

$\lambda_{ij} = \min\{\delta_{ij}, \Omega_{ij}\}$; /* models effect of communication*/

$\lambda_{ij}^{L} = \delta_{ij}^{L}$; /*expected logical arrival rate*/

$\lambda_{.j}^{L'} = \lambda_{.j}^{L'} + \lambda_{ij}^{L}$; /*Equation (3)*/

end_DO

/* Estimate superposition arrival process*/

$MR = 0$; /*temporary variable to represent $\max\{p_{ij}/\lambda_{ij}\}$*/

For *each* input channel to LP_j from LP_i DO

$p_{ij} = \frac{(\lambda_{ij}^{L})}{\lambda_{.j}^{L'}}$; /*Equation (5)*/

$MR = \max\{MR, p_{ij}/\lambda_{ij}\}$; /*Equation (1)*/

end_DO

/*Determine physical superposition arrival rate*/

$\lambda'_{.j} = \frac{1}{MR}$; /*Equation (2)*/

end_SUBROUTINE

Figure 4: Algorithm to estimate arrival process to an LP

SUBROUTINE Estimate_Departure_Process (LP$_j$);

/*Estimates the net departure process from LP$_j$'s output queue */

/*Use Lemmas 1, 2 and 3*/

$\delta^L_{j.} = \delta^{L'}_{j.} = \min\{\lambda^{L'}_{.j}, \mu^L_j\}$;

$\lambda_I = \lambda'_{.j}$;

$\lambda_O = \delta'_{j.} = \min\{\lambda'_{.j}, \mu_j\}$;

$\lambda^L_I = \lambda^{L'}_{.j}$;

$\lambda^L_O = \delta^{L'}_{j.}$;

$p_I = \frac{\lambda^L_I}{(\lambda^L_I + \lambda^L_0)}$;

$p_O = \frac{\lambda^L_O}{(\lambda^L_I + \lambda^L_0)}$;

$\delta_{j.} = \frac{p_O}{\max\{\frac{p_I}{\lambda_I}, \frac{p_O}{\lambda_O}\}}$;

/*Estimate departure process along each output channel*/

For each output arc from LP$_j$ to LP$_k$ DO

$\delta_{jk} = \delta_{j.} \times q_{jk}$; /*departure rate from LP$_j$ to LP$_k$*/

$\delta^L_{jk} = \delta^L_{j.} \times q_{jk}$; /*defined only for $q_{jk} > 0$*/

end_DO

end_SUBROUTINE

Figure 5: Algorithm to estimate departure process from an LP

SUBROUTINE Source_LP_Calculation(LP$_j$);

/*Estimates the arrival process at a source LP*/
/*Estimates are implementation specific to the manner in which messages are generated at source LPs*/
$\lambda_{.j} \leq \mu_j$; /*using algorithm in Figure 7*/
$\lambda_{.j}^L$ /*estimate based on assumptions regarding logical system */

end_SUBROUTINE

Figure 6: Algorithm for calculating arrival and departure processes at source LPs

If messages are waiting then
    process message
else
    generate another message
endif

Figure 7: Algorithm to generate messages at source LPs

service rate at that LP. For our example, $\lambda'_{.1} \leq \mu_1$. We refer to this arrival rate of messages originating from source LPs as the arrival rate of *new* messages to the system.

Assumption 6 At each source LP, the logical arrival rate of messages is equal to the logical service rate at that LP. Thus, $\lambda_{.1}^{L'} = \mu_1^L$.

Assumption 7 For simplicity, we assume that communication time is insignificant. The framework can be easily extended to handle positive communication times by modeling communication channels as single-server, single-queue systems [27].

To begin, we first predict the arrival and departure processes at all LPs with no predecessors. By definition, source LPs have no predecessors. For our example, LP_1 is the source LP (Table II). Let λ_s denote the physical rate at which messages originate from LP_1. As there is only one input channel to LP_1, $\lambda'_{.1} = \lambda_{11} = \lambda_s$ and $p_{11} = 1$, and from Table II and Assumption 6, $\lambda_{.1}^{L'} = \lambda_{11}^L = \mu_1^L = 1$. From Figure 5 and Assumption 5, $\delta'_{1.} = \min\{\lambda'_{.1}, \mu_1\} = \lambda_s$, and $\delta_{1.}^{L'} = \min\{\lambda_{.1}^{L'}, \mu_1^L\} = 1$. Then,

Table III: Estimating Run-Time Performance for Logical System of Figure 1

	Arrival Process From Equations noted in ()						Serv. Proc.		Departure Process From Lemmas noted in ()								
	Frm			(5)	(2)	(3)			(2)		(1)		Max	(2)	To		
LP_j	LP_i	λ_{ij}	λ_{ij}^L	p_{ij}	$\lambda'_{.j}$	$\lambda_{.j}^{L'}$	μ_j	μ_j^L	$\delta'_{j.}$	$\delta_{j.}^{L'}$	p_I	p_O	$\{\frac{p_I}{\lambda_I}, \frac{p_O}{\lambda_O}\}$	$\delta_{j.}$	LP_k	δ_{jk}	δ_{jk}^L
1	1	λ_s	1	1	λ_s	1	μ_1	1	λ_s	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2\lambda_s}$	λ_s	2	$\frac{\lambda_s}{2}$	$\frac{1}{2}$
															3	$\frac{\lambda_s}{2}$	$\frac{1}{2}$
2	1	$\frac{\lambda_s}{2}$	$\frac{1}{2}$	1	$\frac{\lambda_s}{2}$	$\frac{1}{2}$	μ_2	$\frac{1}{4}$	$\frac{\lambda_s}{2}$	$\frac{1}{4}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{4}{3\lambda_s}$	$\frac{\lambda_s}{4}$	4	$\frac{\lambda_s}{8}$	$\frac{1}{8}$
															5	$\frac{\lambda_s}{8}$	$\frac{1}{8}$
3	1	$\frac{\lambda_s}{2}$	$\frac{1}{2}$	1	$\frac{\lambda_s}{2}$	$\frac{1}{2}$	μ_3	$\frac{1}{2}$	$\frac{\lambda_s}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{\lambda_s}$	$\frac{\lambda_s}{2}$	4	$\frac{\lambda_s}{4}$	$\frac{1}{4}$
															5	$\frac{\lambda_s}{4}$	$\frac{1}{4}$
4	2	$\frac{\lambda_s}{8}$	$\frac{1}{8}$	$\frac{1}{3}$													
	3	$\frac{\lambda_s}{4}$	$\frac{1}{4}$	$\frac{2}{3}$													
	—				$\frac{3\lambda_s}{8}$	$\frac{3}{8}$	μ_4	1	$\frac{3\lambda_s}{8}$	$\frac{3}{8}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{4}{3\lambda_s}$	$\frac{3\lambda_s}{8}$	6	$\frac{3\lambda_s}{8}$	$\frac{3}{8}$
5	2	$\frac{\lambda_s}{8}$	$\frac{1}{8}$	$\frac{1}{3}$													
	3	$\frac{\lambda_s}{4}$	$\frac{1}{4}$	$\frac{2}{3}$													
	—				$\frac{3\lambda_s}{8}$	$\frac{3}{8}$	μ_5	1	$\frac{3\lambda_s}{8}$	$\frac{3}{8}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{4}{3\lambda_s}$	$\frac{3\lambda_s}{8}$	6	$\frac{3\lambda_s}{8}$	$\frac{3}{8}$
6	4	$\frac{3\lambda_s}{8}$	$\frac{3}{8}$	$\frac{1}{2}$													
	5	$\frac{3\lambda_s}{8}$	$\frac{3}{8}$	$\frac{1}{2}$													
	—				$\frac{3\lambda_s}{4}$	$\frac{3}{4}$	μ_6	1	$\frac{3\lambda_s}{4}$	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{2}{3\lambda_s}$	$\frac{3\lambda_s}{4}$	—	$\frac{3\lambda_s}{4}$	$\frac{3}{4}$

$\lambda_I = \lambda'_{.1} = \lambda_s$, $\lambda_O = \delta'_{1.} = \lambda_s$, $\lambda^L_I = \lambda^{L'}_{.1} = 1$ and $\lambda^L_O = \delta^{L'}_{1.} = 1$. Therefore, $p_I = \frac{\lambda^L_I}{(\lambda^L_I + \lambda^L_0)} = \frac{1}{2}$, $p_O = \frac{\lambda^L_I}{(\lambda^L_I + \lambda^L_0)} = \frac{1}{2}$, and $\delta_{1.} = \frac{p_O}{\max\{\frac{p_I}{\lambda_I}, \frac{p_O}{\lambda_O}\}} = \frac{p_O}{\max\{\frac{1}{2\lambda_s}, \frac{1}{2\lambda_s}\}} = \frac{1/2}{1/2\lambda_s} = \lambda_s$. Given the branching probabilities in Figure 1, $\delta_{12} = \delta_{1.} \times q_{12} = \lambda_s/2$, and $\delta_{13} = \lambda_s/2$, where q_{ij} is the probability of a message going from LP_i to LP_j.

Next, consider LP_j such that the departure processes of its predecessor LPs have already been estimated. As the arrival process to LP_j is a function of the departure processes of its predecessors, it (the arrival process) can now be estimated. In Figure 1, the arrival process to LP_2 and LP_3 can now be estimated.

Consider LP_2. From Figure 4 and Assumption 7, $\lambda_{12} = \min\{\delta_{12}, \Omega_{12}\} = \delta_{12} = \lambda_s/2$, where Ω_{12} is the communication rate between LP_1 and LP_2. Similarly, $\lambda^L_{12} = \delta^L_{12} = 1/2$. Like LP_1, LP_2 has a single input channel. Thus, $\lambda'_{.2} = \lambda_{12} = \lambda_s/2$ and $\lambda^{L'}_{.2} = 1/2$. From Assumption 4 and Figure 5, $\delta'_{2.} = \min\{\lambda'_{.2}, \mu_2\} = \lambda_s/2$, and $\delta^{L'}_{2.} = \min\{\lambda^{L'}_{.2}, \mu^L_2\} = \min\{1/2, 1/4\} = 1/4$. Then, $\lambda_I = \lambda_s/2$, $\lambda_O = \lambda_s/2$, $\lambda^L_I = 1/2$ and $\lambda^L_O = 1/4$. Therefore, $p_I = \frac{2}{3}$, $p_O = \frac{1}{3}$, and $\delta_{2.} = \frac{1/3}{\max\{\frac{2/3}{\lambda_s/2}, \frac{1/3}{\lambda_s/2}\}} = \frac{1/3}{4/3\lambda_s} = \frac{\lambda_s}{4}$. Clearly, the output waiting rule has an effect on throughput as the departure rate *from* the output queue is less than the arrival rate of messages *into* the output queue. That is, $\delta_{2.} < \delta'_{2.}$.

By considering LPs in order from Table II, the throughput processes from all other LPs can be calculated similarly. As LP_4, LP_5 and LP_6 have multiple (> 1) input channels, the effect of the input waiting rule needs to be calculated to determine the superposition arrival process. For example, LP_4 receives messages from LP_2 and LP_3. From Figure 4, $\lambda_{24} = \delta_{24} = \lambda_s/8$, $\lambda_{34} = \delta_{34} = \lambda_s/4$, $\lambda^L_{24} = \delta^L_{24} = 1/8$ and $\lambda^L_{34} = \delta^L_{34} = 1/4$. Then, $p_{24} = 1/3$ and $p_{34} = 2/3$, and $\max\{\frac{p_{24}}{\lambda_{24}}, \frac{p_{34}}{\lambda_{34}}\} = \frac{8}{3\lambda_s}$. Therefore, $\lambda'_{.4} = \frac{3\lambda_s}{8}$, and $\lambda^{L'}_{.4} = \frac{3}{8}$. The departure process can then be calculated as before. The complete throughput analysis is shown in Table III.

An analysis of the results suggests the following:

- The throughput rate of messages in the system, i.e., the rate at which messages are deleted at the sink LP, is $\frac{3\lambda_s}{4}$, which is less than λ_s, the rate at which new messages enter the system. This indicates a potential bottleneck in the flow of messages in the system.

- At all LPs with multiple input channels, i.e., at LP_4, LP_5 and LP_6, the timestamp increment rate (TSIR) across input channels for an LP is the same. For example, the TSIR for both input channels to LP_4, $\text{C}_{2,4}$ and $\text{C}_{3,4}$, is λ_s (using Equation (4)). As such, the IWR imposes no penalty on throughput. As precedence relationships are always satisfied for LP's with a single input channel, $\lambda'_{.N} = \sum_{i=1}^{n} \lambda'_{iN} = \sum_{i=1}^{n} \lambda_{iN} \ : \forall N$.

- For LP_2, the TSIR for the input queue is less than the TSIR for the output queue. Therefore, the departure rate of messages from the output queue is less than the arrival rate of messages into the output queue. That is, $\delta_{2.} < \delta'_{2.}$. Here, the OWR imposes a penalty on throughput. To decrease the penalty, it is necessary to equalize the TSIR between the input and output queues. At all other LPs, the TSIR in the input and output queues is the same, and so $\delta_{j.} = \delta'_{j.} \ : j \neq 2$.

Clearly, for this system the bottleneck is at LP_2 where the OWR imposes a penalty on throughput. For this LP it can be shown that even if Assumption 4 is violated there may be no degradation in performance. Specifically, as long as $\mu_2 \geq \lambda'_{.2}/2$, the departure rate from LP_2's output queue remains $\delta_{2.} = \lambda_s/4$. At first, this would appear contradictory. That is, we can lower the service rate at LP_2 from $\mu_2 \geq \lambda'_{.2}$ (Assumption 4) to $\mu_2 \geq \lambda'_{.2}/2$, and yet achieve the *same* throughput from the *bottleneck* LP .

At LP_2, the arrival rate into the output queue is $\lambda_s/2$, while the departure rate from the output queue is $\lambda_s/4$. Using Lemma 3, an approximation for $E(N_2)$, the average number of input messages needed before an output message is sent, is 2. Then, as long as a message can be processed at LP_2 before two input messages arrive there, it should have no effect on throughput. This is why as long as $\mu_2 \geq \lambda'_{.2}/2$, there is no effect on throughput.

On the other hand, at LP_3, on average one output message is sent for each input message. That is, $E(N_3) = 1$. So if the service rate decreases to $\mu_3 < \lambda'_{.3}$, then the new departure rate will be less than $\lambda_s/2$, the current departure rate. Therefore, for LP_3 it is important that Assumption 4 be satisfied. In general, as long as $\mu_j \geq \lambda'_{.j}/E(N_j), \ \forall j$, there will be no deterioration in throughput compared to when $\mu_j \geq \lambda'_{.j}, \ \forall j$.

The previous sections have developed a framework to predict run-time performance a priori. The next two sections present experimental validation of the framework.

4 Experimental Analysis

An experimental study was conducted to evaluate the effectiveness of using the framework to predict run-time performance of distributed simulations. The measure of performance was the accuracy of the framework in predicting logical and physical throughput rates. All simulations were performed on an Intel i860 Hypercube with a maximum of 8 processors. The following factors were considered:

- Network Topology: Three different logical systems (logical system 1 given in Figure 1 and logical systems 2 and 3 given in Figure 8) were considered for simulation. The networks (systems 2 and

Logical	Logical Process											
System	1	2	3	4	5	6	7	8	9	10	11	12
1	1	4	2	1	1	1						
2	1	2	1	4	6	8	3	5	1			
3	1	3	4	5	6	7	5	6	7	8	9	1

Table IV: Mean Logical Service Time

3 are identical to those used by Lin and Lazowska [19]) were chosen to simulate different levels of input and output waiting overheads during the simulation.

- Logical System Parameters: Three different logical service time distributions were considered: Uniform, Exponential and Deterministic. Table IV gives the mean logical service time used at each LP for the three logical systems. Parameter values were so chosen that they induced different degrees of input and output waiting overheads during the simulation.

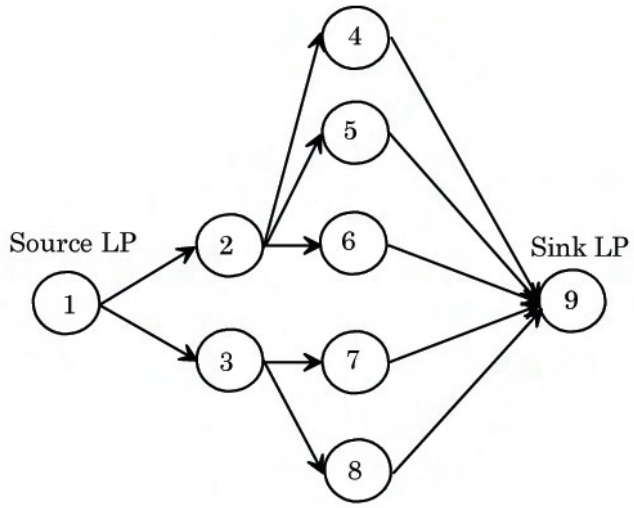
- Routing Probabilities: As in [19], after a message has been received, each output channel had an equal probability of being selected.

- Assignment of LPs to Processors: For each logical system, three *different* assignment of LPs to processors were considered. Care was taken that the three assignments satisfied the same set of assumptions. For example, if we assumed that the service rate was greater than the arrival rate at each LP for a particular assignment of a logical system, then all other assignments for that logical system satisfied the same assumption. Again, the actual values of the arrival and service rates are likely to be different under different assignments. We only presume that the same set of assumptions on the *relative* magnitudes of the arrival and service rates hold true for all assignments for a particular logical system.

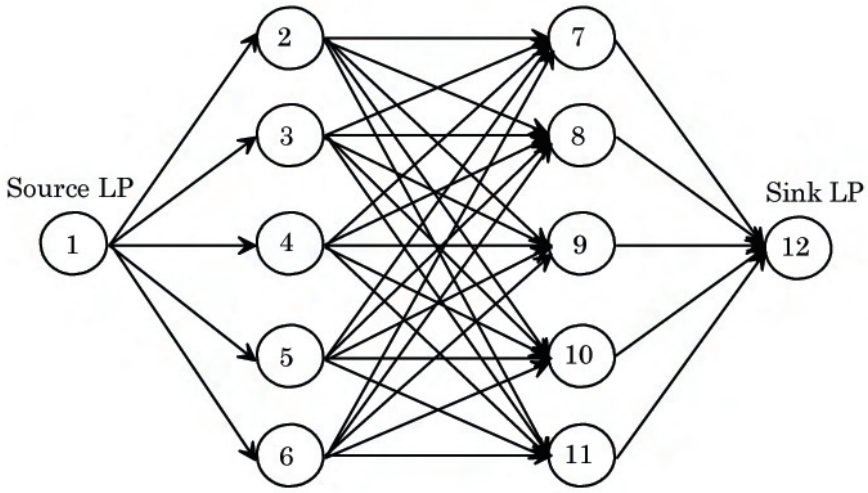
Always, to simulate *steady-state* conditions, data from the simulation was collected after an initial transient period [15], and each experiment was replicated until statistically significant results were obtained at the 90% confidence level.

5 Results

This section discusses the results of the experiments described in the previous section. The results of the experiments are shown in Figure 9 through 12. Only the results using the Uniform distribution



Logical System 2



Logical System 3

Figure 8: Logical systems 2 and 3

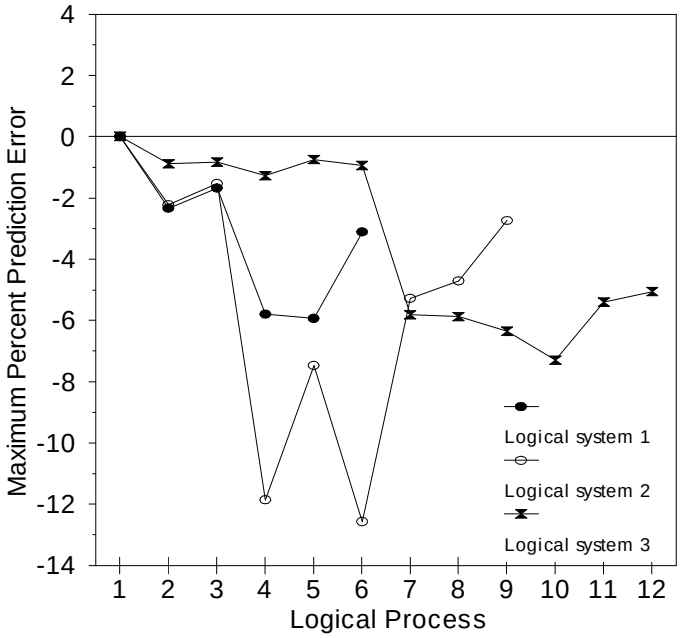


Figure 9: MPPE: Physical arrival rate

for logical service times are shown, as the results using Exponential and Deterministic service times are qualitatively similar.

Figure 9 through 12 show the maximum percent prediction error in using the framework to estimate throughput rates. As mentioned earlier, three different assignment were considered for each logical system. For each assignment, data was gathered on the throughput process. The *percent prediction error* (PPE) at LP_j for a particular assignment is defined as $100 \times [O_j - P_j]/O_j$, where O_j and P_j are the observed and predicted (using the framework) throughput rate from LP_j, respectively. The *maximum percent prediction error* (MPPE) is the maximum of the PPE across the three assignments, and denotes the *worst-case* behavior of the framework. A few points need to be noted:

- As the three assignments for a particular logical system all satisfy the same set of assumptions (given in Section 3), then if our earlier assertion is correct, the observed performance from the three assignments will be similar, and hence the predicted departure rate (using the framework) needs to be computed only once. For example, the different assignments of LPs to processors for logical system 1 (LS_1) (Figure 1) all satisfied Assumptions 4 through 7. The predicted departure rates for each of the assignments will be the same, and is given in Table III.

- As the *actual* values of the departure rates are likely to be different for different assignments of

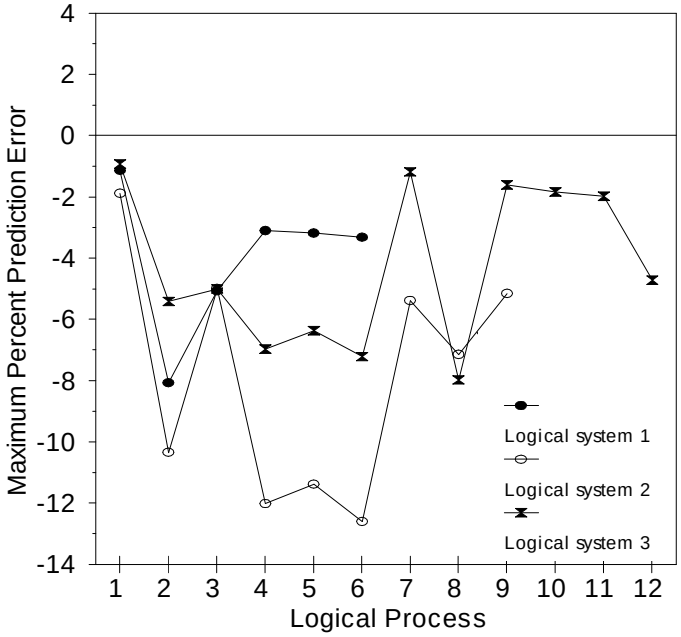


Figure 10: MPPE: Physical departure rate

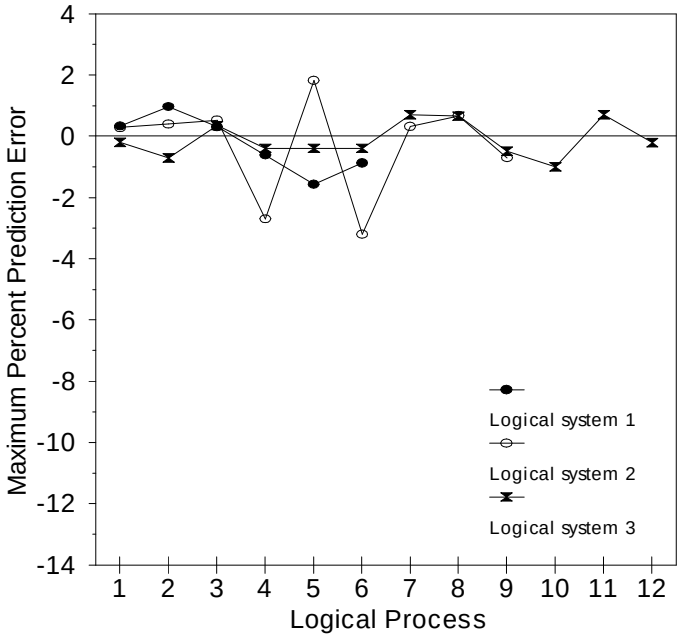


Figure 11: MPPE: Logical arrival rate

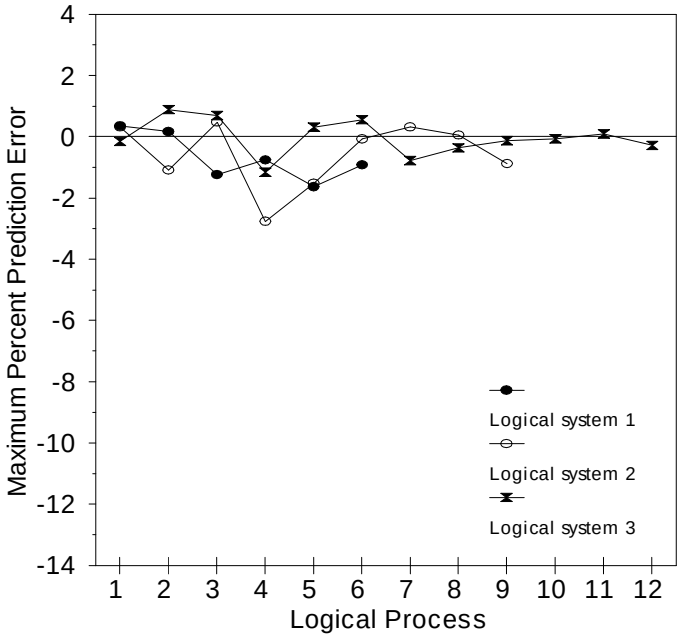


Figure 12: MPPE: Logical departure rate

a logical system, it is necessary to *scale* either the observed, or the predicted values to calculate the percent error. For example in Section 3, the predicted departure from LP_6 for LS_1 is $0.75\lambda_s$. If for a particular assignment, the **observed** value of λ_s is 1, then the predicted departure rate will be 0.75. Then two assignments can be compared simply by knowing the relatives values of λ_s under the two assignments.

Figures 9 and 10 show the MPPE for the physical arrival and departure rates at each LP for the three logical systems (Figures 1 and 8). While the maximum error is around 13% (for LS_2 at LP_6 in Figure 10), in most cases, MPPE is less than 6%. More importantly, in all cases, the MPPE at sink LPs is less than 6%. This suggests that the framework is reasonably accurate in predicting throughput rates from the system, even when *different* mappings of LPs to processors is used. Also, the framework always overestimates the observed throughput rates (Figures 9 and 10). This is expected as the framework ignores many different overheads like interprocessor flow control, message receiving and preparation delays, selecting LPs for processing, etc., in predicting performance. As these overheads have an impact on throughput, the predicted throughput rates will overestimate the observed rates. Thus, the accuracy of the framework is likely to increase with more efficient implementations of the simulation.

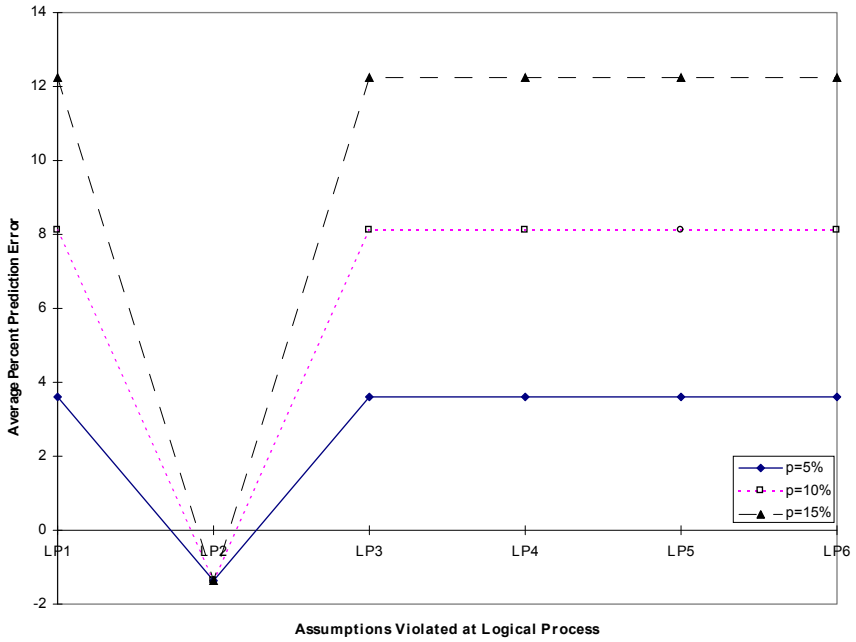


Figure 13: Average PPE: Simulation throughput rate

Figures 11 and 12 show the MPPE for the logical arrival and departures rates. As simulation parameter values are independent of the actual implementation of the simulation, i.e., running the simulation on different computers with the same random numbers should provide identical simulation results but different physical run-time performance, the maximum error for logical time values is much smaller, less than 4% in all cases.

The above experiments show that the framework can predict simulation performance a priori with reasonable accuracy when the assignment of LPs to processors satisfies the same set of assumptions under which the performance was predicted using the framework. Further analysis was done to study the effect of violating these assumptions (Assumptions 1 through 3). Specifically, "how sensitive is the framework to the violation of the these assumptions?"

Logical system 1 (Figure 1) was considered for simulation. The problem specific assumptions under which this system was simulated are given in Assumptions 4 through 7. We now *estimate* the system throughput rate, i.e., the rate at which messages are deleted at sink LPs, using the framework by successively violating Assumption 4 at each LP in the system. Specifically, we first assume that the physical arrival rate λ'_N is greater that the physical service rate μ_N at a particular $\mathrm{LP}_N, 1 \leq N \leq 6$, by a fixed percentage p. The throughput rate is then estimated and the percent prediction error

calculated. Assumption 4 is then violated at a different LP, say LP_K, $K \neq N, 1 \leq K \leq 6$, and the PPE is again calculated. This process is repeated until all LPs in the logical system have been considered. For any one prediction, only one LP violates Assumption 4. Figure 13 shows the average (over the three assignments of LPs to processors) percent prediction error of the system throughput rate for different values of p. Clearly, higher values of p produce larger error, but the average prediction error is always *less* than p. Also, unlike the results of Figure 10, here the framework underestimates the observed throughput rate because we assume the service rate to be lower than it really is in the simulation. When the service rate is underestimated at LP_2, it produces no significant change in predicted throughput rate. This is not the case at other LPs, where an estimation error about the relative magnitude of the physical arrival and service rates produce an error in prediction, though to a smaller degree than p. This agrees with our discussion on Page 22, which shows that the service rate at bottleneck LPs can be reduced without lowering the overall throughput rate, but changes at other LPs have an impact on overall throughput. For this particular logical system, violation of the assumption at all LPs other than LP_2 have an equal impact on prediction error. Therefore, for an accurate prediction, it is necessary that the assumptions of the simulation and the framework match for these LPs. The next section presents the conclusions.

6 Conclusions

This paper has presented a framework to predict distribution simulation run-time performance before the simulation is run. Experimental results suggest that the framework is quite accurate in its prediction.

An important assumption that has been made in predicting performance is that the relative magnitudes of arrival and service rates for a given assignment of LPs to processors can be determined (Assumption 2). While a number of approaches can be used to satisfy the above assumption, including collecting data through exploratory simulations, or estimating values based on the knowledge that in DS the arrival rates are a function of the assignment [27], in many situations for the framework to be useful, it may not be necessary to satisfy the above assumption directly.

For example, when the framework is used in adaptive schemes for task allocation, data on relative arrival and service rates are usually available as part of the simulation [28]. This information can then be used by the framework to predict the performance of a proposed assignment, before the assignment is implemented.

Alternatively, as simulation performance can be predicted for a *given* set of assumptions, it is possible to use the framework to identify conditions that improve simulation performance. For example, as discussed earlier, to improve the performance of LS_1 (Figure 1) efforts should be concentrated at removing the bottleneck condition at LP_2 (see Section 2). Similarly, Page 22 discusses the issue of assigning limited processing power among competing LPs to preserve performance. Additional applications of the framework include evaluating different logical representations of the same physical system, and developing polling systems for DS.

In conclusion, though restricted to feed-forward systems, the framework provides us with a tool to evaluate and improve the cost effectiveness of a DS before the simulation is run, and importantly, in many cases it can be implemented in a manner transparent to the end user.

Acknowledgments

We would like to thank Argonne National Laboratory and LCI, Kent State University, for providing computational support, and two anonymous referees for their helpful comments.

APPENDIX

A Using Lookahead to Improve Performance

Previous studies in conservative simulations have shown that lookahead can reduce the effect of waiting overheads [7, 20, 23]. But, lookahead capabilities incur cost: that of explicitly identifying and implementing it for particular simulations. Thus it is important to use it cost effectively.

Lookahead characterizes the ability of an LP to predict future messages that it will send based on knowledge of messages it has already received. In particular, if an LP has just received a message with timestamp ts, then it can send all output messages with timestamp less than or equal to $ts + d$, where d is the lookahead. To describe the effect of lookahead in the context of our framework, we offer an alternative, but equivalent interpretation. Lookahead can be viewed as a tool that relaxes the effect of the waiting overheads by altering the *relative* logical arrival rate of messages.

Specifically, consider the arrival and departure process at LP_N. Let messages to LP_N arrive at times $1, 2, 3, \cdots$, i.e., with rate $\lambda'_{.N} = 1$, with successive messages having timestamps $1, 2, 3, \cdots$. Let messages to LP_N's output queue be scheduled at times $1, 2, 3, \cdots$, i.e., at rate $\delta'_{N.} = 1$, with successive messages having timestamps $3, 5, 7, \cdots$. Thus, $\lambda^{L'}_{.N} = 1$, and $\delta^{L'}_{N.} = 1/2$. The timestamp increment rate (TSIR) in the input channel is 1, and in the output channel it is 2. An output message is sent whenever its timestamp is less than or equal to the next input message. Here output messages are sent whenever input messages have timestamps $3, 5, 7, \cdots$, which occurs at physical time $3, 5, 7, \cdots$. Therefore, $\delta_{N.} = 1/2 < \delta'_{N.} = 1$.

Now let LP_N have lookahead such that output messages are sent as soon as they are scheduled. Here, output messages are sent whenever input messages have timestamps $1, 2, 3, \cdots$, that is at physical time $1, 2, 3, \cdots$. Therefore, $\delta_{N.} = 1$. Here, to calculate the effective departure rate of messages from the output queue, the OWR sees a logical timestamp rate of $\delta^{L'}_{N.} = 1$, instead of $1/2$ as in the previous case. Then, the TSIR is the same in both the input and output channels as $\lambda'_{.N} = \delta'_{N.} = 1$ and $\lambda^{L'}_{.N} = \delta^{L'}_{N.} = 1$, and therefore the OWR has no effect on throughput.

Lookahead therefore has the potential to change the relative logical timestamp rate in a channel. If the change equalizes the TSIR across the relevant channels, the effect of the waiting rules decrease. For example, in Table III, $\delta^{L'}_{2.} = 1/4$ and $\lambda^{L'}_{.2} = 1/2$, with $\lambda'_{.2} = \delta'_{2.} = \lambda_s/2$, giving $\delta_{2.} = \lambda_s/4$. As the TSIR in the input queue is less than that of the output queue, the OWR has an effect on throughput. If lookahead is introduced at LP_2 to increase the relative logical rate in the output queue, for example

increasing $\delta_{2.}^{L'}$ to 1/2, thereby equalizing the TSIR between the input and output queues, then the effect of the OWR is reduced and $\delta_{2.} = \lambda_s/2$. On the other hand, the TSIR in the input and output queue at LP_3 is the same. Here lookahead will not have any effect on throughput unless it is applied to both the input and output channels. As an example, let lookahead at LP_3 increase the logical arrival rate into the output queue to 2. Then the TSIR in the output queue is $\lambda_s/4$, and in the input queue it is λ_s (as before). As the departure rate from LP_3's output queue is limited by the channel with the least TSIR, using Lemma 1 it can be shown that the departure rate $\delta_{3.}$ will remain unchanged at $\lambda_s/2$. Note that while the departure rate is not affected, the average waiting time in output queue may be reduced by lookahead. And, in systems like closed-queueing networks, this reduction in waiting time can translate to faster throughput. But for feed-forward networks, the effect on throughput will be less pronounced.

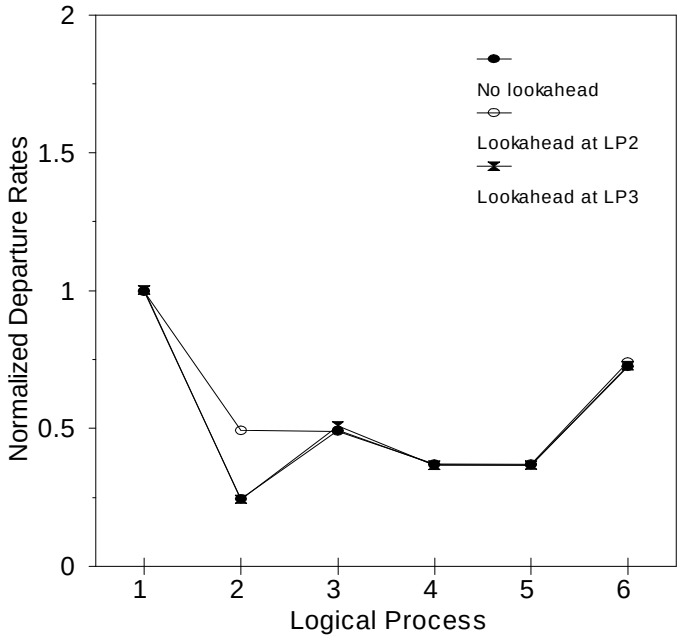


Figure 14: Logical system 1: Effect of lookahead on performance

But now consider the effect of introducing lookahead at LP_2 on the overall performance of the system. From our discussion above, lookahead at LP_2 increases the departure rate to $\lambda_s/2$. This implies that $\lambda_{24} = \lambda_{25} = \lambda_s/4$. Calculating the effect of the IWR at LP_4 and LP_5 we note that $\lambda'_{.4} = \lambda'_{.5} = 3\lambda_s/8$, exactly the same rate as before. Thus, lookahead at LP_2 has only succeeded in pushing the bottleneck from LP_2 to LP_4 and LP_5. Unless we can improve the TSIR along channels

$C_{3,4}$ and $C_{3,5}$, lookahead at LP_2 will have no impact on run-time performance.

Figure 14 shows the effect of introducing lookahead at LP_2 and LP_3 for LS_1. The values have been normalized as it was difficult to generate the same value for λ_s under all three conditions: 1) no lookahead, 2) lookahead at LP_2 and 3) lookahead at LP_3. Note that lookahead at LP_3 has no effect on performance, as discussed above. While, lookahead at LP_2 increases the departure rate from LP_2 to $\lambda_s/2$ (λ_s is normalized to 1), it has no effect on *overall* performance. This is as expected. Therefore the cost of lookahead can be saved without sacrificing performance.

B List of Variable Definitions

q_{ij} = Probability of a message going from LP_i to LP_j.

= $t \to \infty lim$ (fraction of time during the first t time units that messages go from LP_i to LP_j).

p_{ij} = Probability that a message is *chosen* by LP_j from input channel i.

= $t \to \infty lim$ (fraction of time during the first t time units that messages are chosen by LP_j from input channel i)

τ_{ij} = Expected time between arrival of messages from LP_i to LP_j along input channel $\mathrm{C}_{i,j}$.

λ_{ij} = Average arrival rate of messages from LP_i to LP_j along input channel $\mathrm{C}_{i,j}$.

= $1/\tau_{ij}$

τ'_{ij} = Expected time between arrival of messages to LP_j's ready queue from LP_i.

λ'_{ij} = Average arrival rate of messages to LP_j's ready queue from LP_i.

= $1/\tau'_{ij}$

ν_j = Expected service time for a message at LP_j.

(i.e., on the processor to which LP_j is assigned)

μ_j = Average service rate of messages at LP_j.

= $1/\nu_j$

$\psi'_{j.}$ = Expected time between arrival of messages **to** LP_j's *output* queue.

$\delta'_{j.}$ = Average arrival rate of messages **to** LP_j's *output* queue.

= $1/\psi'_{j.}$

ψ_{jk} = Expected time between departure of messages **from** LP_j's *output* queue to LP_k.

δ_{jk} = Average departure rate of messages from LP_j's output queue to LP_k.

= $1/\psi_{jk}$

$\delta_{j.}$ = Average overall departure rate of messages from LP_j's output queue.

= $\sum_k \delta_{jk}$

Ω_{ij} = Average physical communication rate of messages between LP_i and LP_j

= 1/(Expected communication time between LP_i and LP_j).

$E(N_j)$ = Expected number of input messages before an output message is sent from LP_j.

References

[1] S. L. Albin. On poisson approximations for superposition arrival processes in queues. *Management Science*, 28(2):126–137, 1982.

[2] S. L. Albin. Approximating a point process by a renewal process, II: Superposition arrival processes to queues. *Operations Research*, 32(5):1133–1162, 1984.

[3] R. L. Bagrodia and W-T. Liao. Transparent optimizations of overheads in optimistic simulations. In *Proceedings of the Winter Simulation Conference*, 637–645, 1992.

[4] Randal E. Bryant. Simulation of packet communication architecture computer systems. Technical report, Massachusetts Institute of Technology, 1977.

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

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

[7] R. M. Fujimoto. Lookahead in parallel discrete event simulation. In *Proceedings of the International Conference on Parallel Processing*, 3, 34–41, 1988.

[8] R. M. Fujimoto. Performance measurements of distributed simulation strategies. *Transactions of the Society for Computer Simulation*, 6(2):89–132, 1989.

[9] R. M. Fujimoto. Time warp on a shared memory multiprocessor. In *Proceedings of the International Conference on Parallel Processing*, 3, 242–249, 1989.

[10] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):31–53, 1990.

[11] R. M. Fujimoto. Parallel discrete event simulation: Will the field survive? *ORSA Journal on Computing*, 5(3):213–230, 1993.

[12] A. Gafni. Rollback mechanisms for optimistic distributed simulation systems. In *Proceedings of Distributed Simulation*, 61–67. The Society for Computer Simulation, 1988.

[13] D. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.

[14] A. Kumar and R. Shorey. Stability of event synchronization in distributed discrete event simulation. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, 65–72, 1994.

[15] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Book Company, 1982.

[16] Y-B Lin. Understanding the limits of optimistic and conservative parallel simulation. Technical Report 900802, Department of Computer Science and Engineering, University of Washington, August 1990.

[17] Y-B Lin. Effects of waiting overheads on conservative parallel simulation. In *Proceedings of the 25th Annual Simulation Symposium*, 61–70, 1992.

[18] Y-B Lin and E. D. Lazowska. Optimality considerations of time warp parallel simulation. In *Proceedings SCS Multiconference — Distributed Simulation*, 29 – 34. The Society for Computer Simulation, 1990.

[19] Y-B Lin and E. D. Lazowska. On reducing the overheads of conservative parallel simulation without lookahead. *International Journal in Computer Simulation*, 3(3):231–260, 1993.

[20] W. M. Loucks and Preiss B. R. The role of knowledge in distributed simulation. In *Proceedings SCS Multiconference — Distributed Simulation*, 9–16. The Society for Computer Simulation, 1990.

[21] B. D. Lubachevsky. Efficient distributed event-driven simulations of multiple-loop networks. *Communications of ACM*, 32(1):111–123, 1989.

[22] J. Misra. Distributed discrete-event simulation. *Computing Surveys*, 18(1):39–65, 1986.

[23] D. M. Nicol. Parallel discrete-event simulation of FCFS stochastic queueing networks. In *Symposium on Parallel Programming: Experience with Applications, Languages, and Systems*, 124–137, 1988.

[24] D. M. Nicol. Performance bounds on parallel self-initiating discrete-event simulations. *ACM Transactions on Modeling and Computer Simulation*, 1(1):24–50, 1991.

[25] P. Reiher, R. Fujimoto, S. Bellenot, and D. Jefferson. Cancellation strategies in optimistic execution systems. In *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, 112–121. The Society for Computer Simulation, 1990.

[26] M. S. Shanker, W. D. Kelton, and R. Padman. Adaptive distribution of model components via congestion measures. In *Proceedings of the 1989 Winter Simulation Conference*, 640–647, 1989.

[27] M. S. Shanker, W. D. Kelton, and R. Padman. Measuring congestion for dynamic task allocation in distributed simulation. *ORSA Journal on Computing*, 5(1):54–68, 1993.

[28] M. S. Shanker, W. D. Kelton, and R. Padman. Efficient distributed simulation through load balancing. Technical report, Kent State University, 1994.

[29] M. S. Shanker and B. E. Patuwo. The effect of synchronization requirements on the performance of distributed simulations. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, 151–154, 1993.

[30] L. M. Sokol, J. B. Weissman, and P. A. Mutchler. MTW: An empirical performance study. In *Proceedings of the Winter Simulation Conference*, 557–563, 1991.

[31] B. W. Unger and J. G. Cleary. Practical parallel discrete event simulation. *ORSA Journal on Computing*, 5(3):245–248, 1993.

[32] D. B. Wagner. Algorithmic optimizations of conservative parallel simulations. In *Proceedings SCS Multiconference — Distributed Simulation*, 25–32. The Society for Computer Simulation, 1991.

[33] D. B. Wagner and E. D. Lazowska. Parallel simulations of queueing networks: Limitations and potentials. *Performance Evaluation Review*, 17(1):146–155, 1989.

[34] W. Whitt. Approximating a point process by a renewal process, I: Two basic methods. *Operations Research*, 30(1):125–147, 1982.

[35] W. Whitt. The queueing network analyzer. *The Bell System Technical Journal*, 62(9):2779–2815, 1983.

[36] W. Whitt. Performance of the queueing network analyzer. *The Bell System Technical Journal*, 62(9):2817–2843, 1983.