Performance Evaluation of Conservative Algorithms in Parallel Simulation Languages

Rajive L. Bagrodia, *Senior Member*, *IEEE*, and Mineo Takai, *Member*, *IEEE*

Abstract—Parallel discrete event simulation with conservative synchronization algorithms has been used as a high performance alternative to sequential simulation. In this paper, we examine the performance of a set of parallel conservative algorithms that have been implemented in the Maisie parallel simulation language. The algorithms include the asynchronous null message algorithm, the synchronous conditional event algorithm, and a new hybrid algorithm called Accelerated Null Message that combines features from the preceding algorithms. The performance of the algorithms is compared using the Ideal Simulation Protocol. This protocol provides a tight lower bound on the execution time of a simulation model on a given architecture and serves as a useful base to compare the synchronization overheads of the different algorithms. The performance of the algorithms is compared as a function of various model characteristics that include model connectivity, computation granularity, load balance, and lookahead.

Index Terms—Discrete-event simulation, parallel and distributed simulation, conservative algorithms, parallel simulation languages, lookahead, algorithmic efficiency.

1 Introduction

Parallel discrete event simulation (PDES) refers to the execution of a discrete event simulation program on parallel or distributed computers. Several algorithms have been developed to synchronize the execution of PDES models and a number of studies have attempted to evaluate the performance of these algorithms on a variety of benchmarks. A survey of many existing simulation protocols and their performance studies on various benchmarks appears in [11].

A number of parallel simulation environments have also been developed that provide the modeler with a set of constructs to facilitate design of PDES models [5]. One of these is Maisie [3], a parallel simulation language that has been implemented on both shared and distributed memory parallel computers. Maisie was designed to separate the model from the underlying synchronization protocol, sequential or parallel, that is used for its execution. Efficient sequential and parallel optimistic execution of Maisie models have been described previously [2]. In this paper, we evaluate the performance of a set of conservative algorithms that have been implemented in Maisie. The set of algorithms include an asynchronous algorithm (the null message algorithm [8]), a synchronous algorithm (the conditional event algorithm [9]), and a new hybrid algorithm called the Accelerated Null Message (ANM) algorithm that combines the preceding two approaches. Unlike previous performance studies which use speedup or throughput as their metric of comparison, we use $efficiency$ as our primary metric, which is defined using the notion of the Ideal Simulation Protocol, or ISP.

The performance of a parallel simulation model depends on a variety of factors, which include partitioning of the model among the processors, the communication overheads of the parallel platform (both hardware and software overheads), and the overheads of the parallel synchronization algorithm. When a parallel model fails to yield expected performance benefits, the underlying cause is not obvious. For instance, it is difficult to determine whether the problem is due to an inherent lack of parallelism in the model or due to large overheads in the implementation of the synchronization protocol. Research in this area has produced a number of tools to identify the available parallelism in a model. These include analytical tools based on the notion of critical path [1], [17] and experimental tools [16], [25]. In this paper, we use an experimental approach based on the notion of the Ideal Simulation Protocol, or ISP. Tools like ISP allow an analyst to experimentally identify the parallelism that actually exists in a parallel implementation of a simulation model, assuming that the synchronization overhead is zero. In other words, for a specific decomposition of a model, on a given parallel architecture, it is possible to compute a tight lower bound on the execution time of the model. Together, with the measured execution time of the same model with a specific synchronization protocol, this yields the percentage degradation in performance due to the simulation algorithm, which directly translates into a measure of the relative efficiency of the synchronization scheme. Thus, ISP may be used to compute the efficiency of a given synchronization algorithm and provide a suitable reference point to compare the performance of different algorithms, including conservative, optimistic, and adaptive techniques.

• *The authors are with the Computer Science Department, University of California, Los Angeles, CA 90095. E-mail: {rajive, mineo}@cs.ucla.edu.*

Manuscript received 16 Oct. 1995; revised 2 June 1998; accepted 31 Aug. 1999.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 100004.

1045-9219/00/$10.00 © 2000 IEEE

The remainder of the paper is organized as follows: Section 2 describes the conservative algorithms that have been used for the performance study reported in this paper. Section 3 describes the Ideal Simulation Protocol and its use in separating protocol-dependent and independent overheads. Section 4 describes implementation issues of synchronization algorithms, including language level constructs to support conservative algorithms. Section 5 presents performance comparisons of the three conservative algorithms with the lower bound prediction of ISP. Related work in the area is described in Section 6. Section 7 is the conclusion.

2 Conservative Algorithms

In parallel discrete event simulation, the physical system is typically viewed as a collection of physical processes (PPs). The simulation model consists of a collection of Logical Processes (LPs), each of which simulates one or more PPs. The LPs do not share any state variables. The state of an LP is changed via messages which correspond to the events in the physical system. In this section, we assume that each LP knows the identity of the LPs that it can communicate with. For any LP_p, we use the terms $dest\text{-}set_p$ and $source\text{-}set_p$ to respectively refer to the set of LPs to which LP_p sends messages and from which it receives messages.

The causality constraint of a simulation model is normally enforced in the simulation algorithm by ensuring that all messages to a Logical Process (LP) are processed in an increasing timestamp order. Distributed simulation algorithms are broadly classified into *conservative* and *optimistic* based on how they enforce this. Conservative algorithms achieve this by not allowing an LP to process a message with timestamp t until it can ensure that the LP will not receive any other message with a timestamp lower than t. Optimistic algorithms, on the other hand, allow events to be potentially processed out of timestamp order. Causality errors are corrected by rollbacks and recomputations. In this paper, we focus on the conservative algorithms.

To simplify the description of the algorithms, we define the following terms. We assume that the communication channels are FIFO.

- **Earliest Input Time**(EIT_p). EIT_p is a lower bound on the timestamp of any future message that LP_p may *receive*.
- **Earliest Output Time**(EOT_p). EOT_p is a lower bound on the timestamp of any future message that LP_p may *send*.
- **Earliest Conditional Output Time**($ECOT_p$). $ECOT_p$ is a lower bound on the timestamp of any future message that LP_p may *send assuming that LP_p will not receive any more messages*.
- **Lookahead**(la_p). la_p is a lower bound on the duration after which the LP will send a message to another LP.

The value of EOT and ECOT for a given LP depends on its EIT, the unprocessed messages in its input queue, and its lookahead. Fig. 1 illustrates the computation of EOT and $ECOT$ for an LP that models a FIFO server. The server is assumed to have a minimum service time of one time unit, which is also its lookahead. The three scenarios in the figure respectively represent the contents of the input message queue for the LP at three different points in its execution; messages already processed by the LP are shown as shaded boxes. Let T_{next} refer to the earliest timestamp of a message in the input queue. In Fig. 1a, since $T_{next} = 22 < EIT = 30$, both EIT and $ECOT$ are equal to T_{next} plus the minimum service time. In Fig. 1b, EOT is equal to $EIT = 30$ plus the minimum service time because the LP may still receive messages with a timestamp less than $T_{next} = 40$, but no smaller than EIT. However, the $ECOT$ is equal to T_{next} plus minimum service time because if the LP does not receive any more messages, its earliest next output will be in response to processing the message with timestamp T_{next}. In Fig. 1c, $T_{next} = \infty$ because there are no unprocessed messages. Therefore, $ECOT$ is equal to ∞.

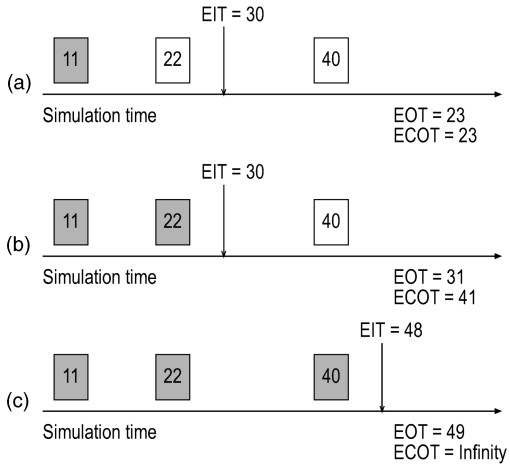

Fig. 1. The computation of EOT and ECOT: The LP is modeling a server with a minimum service time of one unit. Shaded messages have been processed by the LP.

Various conservative algorithms differ in how they compute the value of EIT for each LP in the system. By definition, in a conservative algorithm, LP_p can only process messages with timestamp $\leq EIT_p$. Therefore, the performance of a conservative algorithm depends on how efficiently and accurately each LP can compute the value of its EIT. In the following sections, we discuss how three different algorithms compute EIT.

2.1 Null Message Algorithm

The most common method used to advance EIT is via the use of *null* messages. A sufficient condition for this is for an LP to send a *null* message to every LP in its *dest_set*, whenever there is a change in its EOT, assuming that message arrival and retrieval are performed in FIFO manner. Each LP computes its EIT as the *minimum of the most recent EOT received* from every LP in its *source-set*. Note that a change in the EIT of an LP typically implies that its EOT will also advance. A number of techniques have been proposed to reduce the frequency with which null messages are transmitted; for instance, null messages may be piggybacked with regular messages or they may be sent only when an LP is blocked rather than whenever there is a change in its EOT [24], [19].

The null message algorithm requires that the simulation model not contain zero-delay cycles: If the model contains a cycle, the cycle must include at least one LP with positive lookahead (i.e., if the LP accepts a message with timestamp t, any message generated by the LP must have a timestamp that is strictly greater than t) [19].

2.2 Conditional Event Algorithm

In the conditional event algorithm [9], the LPs alternate between an *EIT computation phase* and an *event processing* phase. For the sake of simplicity, we first consider a synchronous version which ensures that no messages are in transit when the LPs reach the *EIT computation phase*. In such a state, the value of EIT for an LP_p is equal to the minimum of ECOT over all the LPs in the transitive closure[1] of $source\text{-}set_p$.

The algorithm can be made asynchronous by defining the EIT to be the minimum of all ECOT values for the LPs in the transitive closure of the *source-set* and the timestamps of all the messages in transit from these LPs. Note that this definition of EIT is the same as the definition of Global Virtual Time (GVT) in optimistic algorithms. Hence, any of the GVT computation algorithms [6] can be used. Details of one such algorithm are described in Section 4.4.

2.3 Accelerated Null Message Algorithm

We superimpose the null message protocol on an asynchronous conditional event protocol which allows the null message protocol to perform unhindered [15]. The EIT for any LP is computed as the maximum of the values computed by the two algorithms. This method has the potential of combining the efficiency of the null message algorithm in the presence of good lookahead with the ability of conditional event algorithm to execute even without lookahead—a scenario in which the null message algorithm alone will deadlock. In models with poor lookahead, where it may take many rounds of null messages to sufficiently advance the EIT of an LP, ANM could directly compute the earliest global event considerably faster.

3 Isolating Synchronization Overheads

Most experimental performance studies of parallel simulations have used speedup or throughput (i.e., number of events executed per unit time) as the performance metric on the effectiveness of a particular protocol for improving performance. While both metrics are appropriate for evaluating benefits from parallel simulation, they do not shed any light on the efficiency of a simulation protocol.

A number of factors affect the execution time of a parallel simulation model. We classify the factors into two categories—*protocol-independent* factors and *protocol-specific* factors. The former refers to the hardware and software characteristics of the simulation engine, like the computation speed of the processor, communication latency, and cost of a context switch, together with model characteristics, like partitioning and LP-to-processor mapping, that determine the inherent parallelism in the model. The specific simulation protocol that is used to execute a model has relatively little control on reducing the overhead contributed by these factors. In contrast, the overhead due to the protocol-specific factors does depend on the specific simulation protocol that is used for the execution of a model. For conservative protocols, these may include the overhead of processing and propagating null messages [8] or conditional event messages [9], and the idle time of an LP that is blocked because its EIT has not yet advanced sufficiently to allow it to process a message that is available in its input queue. In the case of optimistic protocols, the *protocol-specific* overheads may include state saving, rollback, and the Global Virtual Time (GVT) computation costs.

A separation of the cost of the protocol-specific factors from the total execution time of a parallel simulation model will lend considerable insight into its performance. For instance, it will allow the analyst to isolate a model where performance may be poor due to lack of inherent parallelism in the model (something that is not under the control of the protocol) from one where the performance may be poor due to a plethora of null messages (which may be addressed by using appropriate optimizations to reduce their count). Two primary approaches have been used for this purpose: analytical tools that are typically based on critical path analyses and experimental approaches that measure the actual overheads in an implementation. In the past, critical path analyses have been used to prove theoretical lower bounds and related properties [1], [17] of a parallel simulation model. The execution time of all events in the critical path of a model is summed together with estimates of various overheads due to message communication, entity scheduling, and other factors to yield an analytical formula to compute the lower bound. The tools may use a purely probabilistic model to estimate the various costs or derive the formulas based on appropriately aggregated measurements of the known sources of overhead. The primary drawback of the analytical models is that the computed bound may not be sufficiently tight. In this section, we describe how ISP may be used to experimentally compute a tight lower bound on the execution time of the model on a given parallel architecture.

The primary idea behind ISP is simple: The model is executed once and a trace of the messages accepted by each LP is collected locally by the LP. In a subsequent execution, an LP may simply use the trace to *locally* deduce when it is safe to process an incoming message; no synchronization protocol is necessary. The resulting lower bound predicted by ISP is realistic because it accounts for all the protocol-independent factors that must be incurred by any simulation protocol. These factors include overheads that are hard to estimate accurately like communication latencies for messages that must be sent regardless of the protocol, event list management, and others. Also, because the execution time is measured, all protocol-independent overheads are directly included in the execution time, and there is no likelihood that small overheads that may together make a substantial contribution to the execution time of the model are ignored. There is a potential for the ISP to perturb the execution time of the model by introducing its own

1. Transitive closure of *source-set* in many applications where entities are linked bidirectionally is identical to the set of *all* LPs in the system.

protocol-specific overheads, but, as shown by the measurements reported in Section 4.6, these overheads are relatively insignificant. Given this lower bound, it is easy to measure the efficiency of a protocol as described in Section 5.

Besides serving as a reference point for computing the efficiency of a given protocol, a representative presimulation with ISP can yield a realistic prediction of the available parallelism in a simulation model. If the speedup potential is found to be low, the user can modify the model, which may include changing the partitioning of the system, changing the assignment of LPs to processors, or even moving to a different parallel platform, in order to improve the parallelism in the model.

4 Implementation Issues

The performance experiments were executed using the Maisie simulation language. Each algorithm, including ISP, was implemented in the Maisie runtime system. A programmer develops a Maisie model and selects among the available simulation algorithms as a command line option. This separation of the algorithm from the simulation model permits a more consistent comparison of the protocols than one where the algorithms are implemented directly into the application. In this section, we briefly describe the Maisie language and the specific constructs that have been provided to support the design of parallel conservative simulations. The section also describes the primary implementation issues for each algorithm.

4.1 Maisie Simulation Environment

Maisie [3] is a C-based parallel simulation language, where each program is a collection of entity definitions and C functions. An entity definition (or an entity type) describes a class of objects. An entity instance, henceforth referred to simply as an entity, represents a specific LP in the model—entities may be created dynamically and recursively.

The events in the physical system are modeled by message communications among the corresponding entities. Entities communicate with each other using buffered message passing. Every entity has a unique message buffer; asynchronous send and receive primitives are provided to deposit and remove messages from the buffer, respectively. Each message carries a timestamp which corresponds to the simulation time of the corresponding event. Specific simulation constructs provided by Maisie are similar to those provided by other process-oriented simulation languages. Details of basic Maisie constructs are described in [3]. In addition, Maisie provides constructs to specify the communication topology of a model and the lookahead properties of each entity. These constructs were used to investigate the impact of different communication topologies and lookahead patterns on the performance of each of the conservative algorithms described earlier.

Fig. 2 is a Maisie model for an FCFS server. This piece of code constructs the tandem queue system described in Section 5 and is used, with minor changes, in the experiments described there. The entity "Server" defines a message type "Job" which is used to simulate the requests for service to the server. The body of the entity consists of an unbounded number of executions of the **wait** statement

```
1  entity Server{MeanTime, PrevServer, NextServer}
2      clocktype MeanTime;
3      ename PrevServer;
4      ename NextServer;
5  {
6      message Job{};
7      clocktype JobServiceTime = ShiftedExpon(MeanTime);
8      add_source(PrevServer);
9      add_dest(NextServer);
10     while (True) {
11         setlookahead(JobServiceTime, CLOCKTYPE_MAX);
12         wait until mtype(Job) {
13             hold(JobServiceTime);
14             invoke NextServer with Job{};
15             JobServiceTime = ShiftedExpon(MeanTime);
17         }
18     }
19 }
```

Fig. 2. Maisie entity code for "First Come First Served" (FCFS) server with user defined lookahead.

(line 12). Execution of this statement causes the entity to block until it receives a message of the specified type (which in this case is "Job"). On receiving a message of this type, the entity suspends itself for "JobServiceTime" simulation time units to simulate servicing the job by executing the **hold** statement (line 13). The semantics of the hold statement guarantee that any messages received by the entity in the interim are implicitly buffered in the entity's message queue and retrieved by the entity subsequently. When the (simulation) time specified by the hold statement has expired, the entity forwards the job to the adjacent server identified by variable "NextServer," using the **invoke** statement (line 14).

The performance of many conservative algorithms depends on the connectivity of the model. In the absence of any information to the contrary, the algorithm must assume that every entity belongs to the source-set and dest-set of every other entity. However, Maisie provides a set of constructs that may be used by an entity to add or remove elements from these sets: **add_source**(), **del_source**(), **add_dest**(), and **del_dest**(). In Fig. 2, the server entity specifies its connectivity to the other entities using the **add_source**() and **add_dest**() functions (lines 8 - 9).

Lookahead is another important determinant of performance for conservative algorithms. Maisie provides a predefined function **setlookahead**() that allows the user to specify its current lookahead to the run-time system. The run-time system uses this information to compute a better estimate of EOT and ECOT than may be feasible otherwise. For instance, when the server is idle, it is possible for the server to precompute the service time of the next job (line 15), which becomes its lookahead and is transmitted to the run-time system using the function **setlookahead**() (in line 11) before it suspends itself. The run-time system detects errors caused by the specification of inaccurate lookahead values that result in an entity generating messages with timestamps that are smaller than the EOT or ECOT value for an entity.

4.2 Entity Scheduling

If multiple entities are mapped to a single processor, a scheduling strategy must be implemented to choose one

among a number of entities that may be eligible for execution at a given time. An entity is *eligible* to be executed if its input queue contains a message with a timestamp that is lower than the current EIT of the entity. In the Maisie runtime, once an entity is scheduled, it is allowed to execute as long as it remains eligible. This helps reduce the overall context switching overhead since more messages are executed each time the entity is scheduled, but may result in the *starvation* of other entities mapped on the same processor. This, in turn, may cause blocking of other processors that might be waiting for messages from these entities, resulting in suboptimal performance. Note, however, that without prior knowledge of the exact event dependencies, no scheduling scheme can be optimal.

An alternative scheduling strategy, used in the Global Event List algorithm, is to schedule events across all entities mapped to a processor in the order of their timestamps. We examine and discuss the overheads caused by these two scheduling strategies in Section 5.

4.3 Null Message Algorithm

One of the tunable parameters for any null message scheme is the frequency with which null messages are transmitted. Different alternatives include eager null messages—an LP sends null messages to all successors as soon as its EOT changes, lazy null messages—an LP sends null messages to all successors only when it is idle, or demand driven—an LP sends null message to a destination only when the destination demands to know the value of its EOT.

The performance of different null message transmission schemes, including eager, lazy, and demand driven schemes, has been discussed in [24]. The experiments found that, for the set of benchmarks included in that study, demand driven schemes performed poorly, whereas the lazy null message scheme combined with eager event sending (i.e., regular messages sent as soon as they are generated, as is the case in our runtime) was found to marginally outperform other schemes. We use the lazy null message scheme in our experiments. This has the advantage of reducing the overall number of null messages because several null messages may be replaced by the latest message. However, the delay in sending null messages may delay the processing of real messages at the receiver.

4.4 Conditional Event Computation

For the conditional event and the ANM protocol, it is necessary to periodically compute the earliest conditional event in the model. As discussed in Section 2.3, an asynchronous algorithm allows the earliest event time to be computed without the need to freeze the computation at each node.

In our runtime system, this computation takes place in phases. A new phase of a processor starts when the processor has finished processing all the messages with timestamps less than or equal to the global ECOT for the current phase. At the start of a new phase i, the jth processor computes its E_{ij} and M_{ij}. E_{ij} represents the ECOT of the jth processor in phase i, which is defined to be the minimum ECOT value among all entities mapped on processor j. M_{ij} is the smallest timestamp that is sent by the processor in phase $i-1$. M_{ij} accounts for the messages in transit; in the worst case, none of the messages sent by the processor in phase $i-1$ may have been received and accounted for in computing the E_i of other processors. Thus, the global ECOT for phase i is calculated as follows:

$$\min_{j=1\ldots n}\{\min\{E_{ij}, M_{ij}\}\},$$

where n is the number of processors. No message sent during phases $i-2$ and earlier is in transit because a new phase starts only when all messages from the previous phase have been received. This, along with the FIFO communication assumption, implies that E_{ij} takes into account all messages sent to the processor by any processor during phase $i-2$.

In phase i, every processor sends an ECOT message which contains its E_i and M_i and can compute the global ECOT to be the minimum of ECOT messages from all the processors. As each processor summarizes ECOT information for all entities stored on it, the number of messages required for a global ECOT computation is reduced from the square of the number of entities to the square of the number of processors. Note that the ECOT message needs to carry the phase identifier with it. But, since a new phase does not start until the previous one has finished, a Boolean flag is sufficient to store the phase identifier.

4.5 Accelerated Null Message Algorithm

The primary motivation behind the ANM protocol was to accelerate the computation of EIT by using the conditional event protocol for models that contain cycles with low lookahead values. The protocol uses null messages as described previously in Section 4.3; the global ECOT computation is initiated at a processor only when the processor has no regular messages to process. The behavior of the ANM protocol is similar to that of the NULL protocol when null messages successfully update the EIT of each entity. However, updating individual EIT values may sometimes delay the initiation of the global ECOT computations at some processors, resulting in growth of null messages, even in cases where the COND protocol outperforms the NULL protocol. We will examine the performance correlation between the three algorithms under various conditions in Section 5.

4.6 ISP

The Ideal Simulation Protocol (ISP) has been implemented as one of the available simulation protocols in the Maisie environment. The implementation uses the same data structures, entity scheduling strategy, message sending and receiving schemes as the conservative algorithms discussed earlier.

The primary overhead in the execution of a model with ISP is the time for reading and matching the event trace. The implementation minimizes this overhead as follows: The entire trace is read into an array for each entity at the beginning of the simulation to exclude the I/O time for reading the trace file from the measurements. The primary overhead for ISP is the time to check if an incoming message "matches" the next message from its trace. ISP implements this check using the unique numeric identifier that is assigned to each message by the runtime system to

TABLE 1
Typical Costs for Primitive Simulation Operations Measured on a SPARCserver 1000

Numeric comparison in ISP	0.99 [μs]
Transmission time for a local message	6.91 [μs]
Transmission time for a remote message	17.23 [μs]
Entity scheduling time in a model with 2 entities	8.33 [μs]
Entity scheduling in a model with 64 entities	13.79 [μs]

implement the message ordering semantics defined by Maisie. Thus, the only "synchronization" overhead in executing a simulation model with ISP is the time required to execute a numeric comparison operation for each incoming message and every context switch to determine which message is to be processed next. Table 1 lists the overhead of numeric comparison in ISP on a SPARCserver 1000. For comparison, the table also lists the transmission time for a local message defined as the time required to send a message from the data space of one entity to another entity scheduled on the same processor, the transmission time for a remote message, and the average entity scheduling time for two different model sizes. The entity scheduling time includes the queue management and context switching times, where the entities are stored in a sorted queue implemented as a splay tree. These measurements clearly show that the overhead associated with ISP is very low and is negligible when compared with other overheads of parallel simulation. The average computation granularity in the CQNF benchmark used for the performance study reported in this paper is 51.48 microseconds, which implies that the additional overhead for processing each message in ISP is less than 2 percent even for a benchmark with such a low computation granularity. Relatively, this overhead becomes even smaller when compared with the execution time of a parallel program which includes other overhead factors, like remote message communication time and idle time at a processor, that are not related to the ISP overhead. Also, note that the difference in entity scheduling times for the case with 2 and 64 entities is almost a factor of 2, which is an indication of the increased accuracy of the predictions with ISP as compared with other approaches that use approximations of these factors.

5 Experiments and Results

Both sequential and parallel implementations of the programs used for the study were written in Maisie. The parallel versions differed from the sequential ones only in that the former included the following additional directives: 1) explicit assignment of Maisie entities to processors, 2) code to create the source and destination sets for each entity, and 3) specification of lookahead. The measurements were taken on a SPARCserver 1000 with eight SuperSPARC processors, and 512M of shared memory.

We selected the Closed Queuing Networks (CQN), a widely used benchmark, to evaluate parallel simulation algorithms. Although Maisie has been used to simulate many real-world applications [4], [10], a synthetic benchmark was used in this paper as it allows the performance of

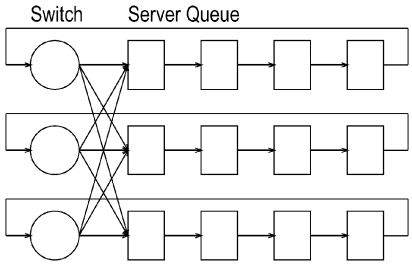

Fig. 3. A Closed Queuing Network $(N = 3, Q = 4)$.

the algorithms to be compared as a function of significant model characteristics like communication topology, computation granularity, and lookahead in a controlled manner. These characteristics are hard to modify in models of real world applications as they are an inherent property of the corresponding system.

The CQN model can be characterized by the following parameters:

- The type of servers and classes of jobs. Two separate versions of this model were used: CQNF, where each server is FIFO, and CQNP, where each server is a priority-preemptible server.
- The number of switches and tandem queues (N) and the number of servers in each tandem queue (Q).
- Number of jobs that are initially assigned to each switch, J/N.
- Service time. The service time for a job is sampled from a shifted-exponential distribution with mean value of S unit time (indicating a mean value of $S-1$ time unit plus 1). The choice of the shifted-exponential distribution ensures that the minimum lookahead for every entity is nonzero, thus preventing a potential deadlock situation with the null message algorithm.
- Topology. When a job arrives at a switch, the switch routes the job to any one of the N tandem queues with equal probability after 1 time unit. After servicing a job, each server forwards the job to the next server in the queue; the last server in the queue must send the job back to its unique associated switch.
- The total simulation time, H

Each switch and server is programmed as a separate entity. Thus, the CQN model has a total of $N(Q+1)$ entities. The topology of the network, for $N = 3$ and $Q = 4$, is shown in Fig. 3. Henceforth, we use the term *stage* to refer to the switch and its associated tandem queue of servers. The service times for the servers are precomputed as shown in Fig. 2 for all experiments except those in Section 5.3.1 that examine the impact of lookahead on model performance.

The performance of parallel algorithms is typically presented by computing speedup, where

$$\text{Speedup}(P) = \frac{\text{Sequential execution time}}{\text{Parallel execution time on } P \text{processors}}.$$

However, as we will describe in the next section, the sequential execution time of a model, using the standard Global Event List (GEL) algorithm [19], does not provide

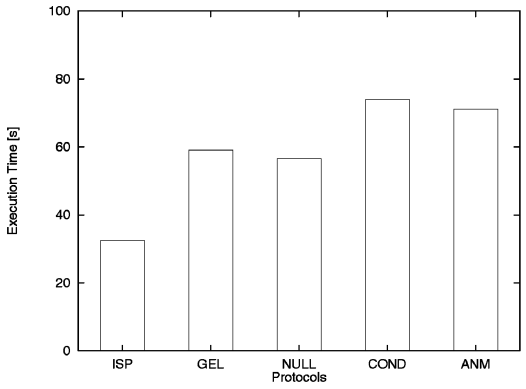

Fig. 4. Execution times of a CQNF model on one processor.

the lower bound for single processor execution and can in fact be slower than a parallel conservative algorithm. For this reason, the speedup metric as defined above might show super-linear behavior for some models or quickly become saturated if the model does not have enough parallelism. In this paper, we compare the algorithms by computing the efficiency of each algorithm using the execution time with ISP as the base because ISP provides a tight lower bound on the execution time of the model. The efficiency of a protocol S on architecture A is defined as follows:

$$\text{Efficiency}(S, A) = \frac{\text{Execution time using ISP on A}}{\text{Execution time using Protocol } S \text{ on } A}.$$

As defined, the efficiency of each protocol identifies the fraction of the execution time that is devoted to synchronization-related factors or the *protocol-specific* operations in the execution of a model.

We begin our experiments (Section 5.1) by comparing the *single processor* execution time of each of the five protocols: GEL(global event list), NULL(null message), COND(conditional event), ANM, and ISP. This configuration is instructive because it identifies the unique set of "sequential" overheads incurred by each protocol. In subsequent sections, we investigate the impact of modifications in various model characteristics like model connectivity (Section 5.2), lookahead properties (Section 5.3), load balancing (Section 5.4), and computation granularity (Section 5.5).

5.1 Simulation of a CQNF Model on One Processor

Fig. 4 shows the execution times for a CQNF model with $N = 32$, $Q = 3$, $J = 1,024$, $S = 10$, and $H = 100,000$ on one processor of the SPARCserver 1000. As seen in the figure, the execution times of the protocols differ significantly even on one processor. The computation granularity associated with each event in our model is very small, which makes the protocol-specific overheads associated with an event relatively large. This allows us to clearly compare these overheads for the various protocols, which is the primary purpose of the study.

In Fig. 4, ISP has the best execution time which can be regarded as a very close approximation of the *pure* computation cost for the model. We first compare the ISP and GEL protocols. The major difference between the two protocols is the entity scheduling strategy. As described in Section 4.6, ISP employs the same entity scheduling scheme as the conservative protocols, where each entity removes and processes as many safe messages from its input queue as possible before being swapped out. For ISP, this means that once an entity has been scheduled for execution, it is swapped out only when the *next* message in its local trace is not available in the input queue for that entity. In contrast, the GEL algorithm schedules entities in strict order of message timestamps, which can cause numerous context switches. Table 2 shows the total number of context switches for each protocol for this configuration. The number of context switches for ISP is approximately 15 times less than that required by the GEL algorithm. Considering that a total of 1,200,000 messages were processed, the GEL protocol requires almost one context switch per message.[2]

TABLE 2
Number of Context Switches, Null Messages and Global ECOT Computations in One Processor Executions $[\times 1,000]$.

	ISP	GEL	NULL	COND	ANM
Context Switches	85	1,257	393	1,149	914
Null Messages	0	0	1,732	0	1
Global Computation	0	0	0	49	10

The number of regular messages processed in the simulation is 1,193 $[\times 1,000]$.

Further, the GEL algorithm is implemented in the Maisie run-time system using a splay tree to efficiently sort entities in the order of their earliest messages. However, the other protocols can use the more efficient unordered list because the entities are scheduled on the basis of safe messages rather than in the strict order of message timestamps. Thus, the performance difference between the ISP and GEL protocols is primarily due to the differences in their costs for context switching and entity queue management. The context switching overheads are particularly relevant to the process-oriented paradigm that is used by Maisie.

We next compare the performance of ISP and the three conservative protocols. First, ISP has fewer context switches than the other protocols because the number of messages that can be processed by an entity using the conservative protocols is bounded by its EIT, whereas ISP has no such upper bound. The differences in the number of context switches among the three conservative protocols are due to differences in their computation of EIT, as well as due to the additional context switches that may be required to process synchronization messages. For instance, the COND protocol needs to process all the messages with timestamps less than the global ECOT in the model before computing the next global ECOT, which incurs many context switches for every global ECOT computation. Therefore, it has three times the number of context switches than the NULL protocol, which does not require a context switch to process every null message. The CQNF model used in this experiment consists of 128 entities and each switch entity has a lookahead of one time unit, thus the window size between two global ECOT computations is small. As shown in Table 2, the COND protocol had approximately 49,000 global computations,

2. Since the Maisie runtime system executes as a separate "main" thread, the number of context switches can be larger than the total number of messages.

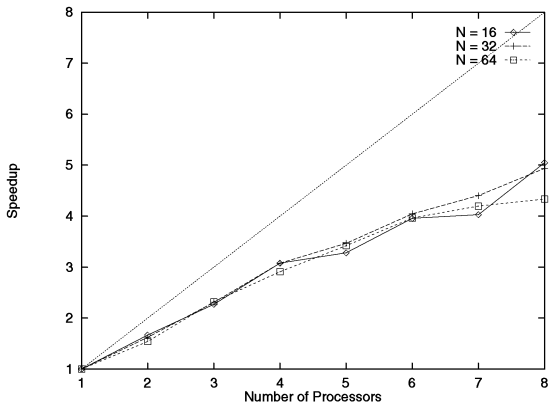

Fig. 5. CQNF: Speedup achieved by ISP.

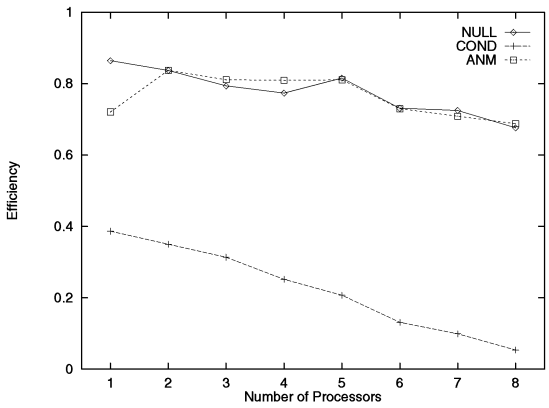

Fig. 6. CQNF: Protocol Efficiency for $N = 16$.

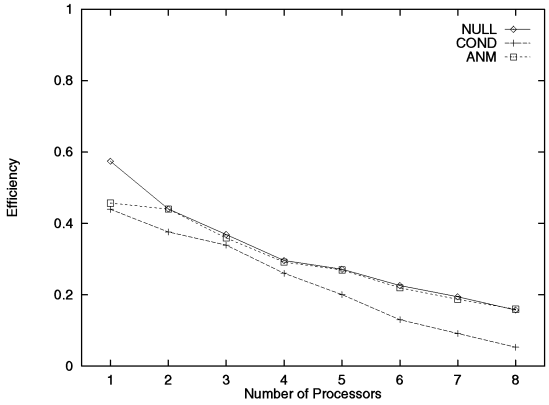

Fig. 7. CQNF: Protocol Efficiency for $N = 32$.

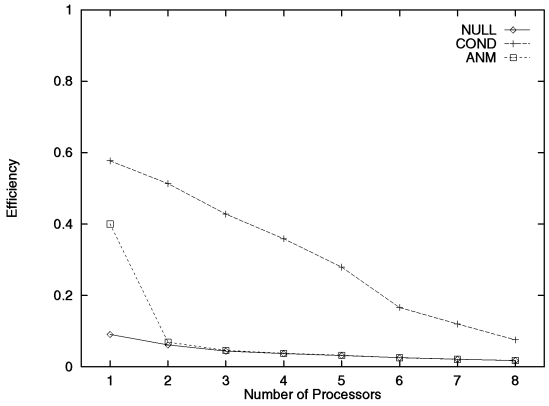

Fig. 8. CQNF: Protocol Efficiency for $N = 64$.

which averages to one global computation for every advance of two units in simulation time.

Even though the NULL protocol uses few context switches, its execution time is significantly longer than ISP because of the large number of null messages (1,732,000 from Table 2) that must be processed and the resulting updates to EIT. The execution time using the ANM protocol is between those of the NULL and COND protocols. The ANM protocol requires fewer null messages because of its use of the ECOT computations. However, such global computations incur more context switches, resulting in longer execution time than the one using only null messages.

5.2 Communication Topology

In Section 4.1, we introduced Maisie constructs to alter the default connectivity of each entity. As the next experiment, we examine the impact of communication topology on the performance of parallel conservative simulations.

For each protocol, the CQNF model was executed with different values of $N = \{16, 32, 64\}$; the total number of entities ($N(Q+1)$) is kept constant among the different configurations to also keep the size of the model constant, i.e., $N(Q+1) = 128$. The other parameters for the model were set to $J = 1,024$, $S = 10$, and $H = 100,000$. As N increases, the connectivity of the model increases dramatically because each of the N switch entities has N outgoing channels, while each of the Q server entities in each tandem queue has only one outgoing channel. For instance, the spatial locality, which is defined as the total number of links in the model divided by the number of links to completely connect all the entities, is 0.03, 0.07, and 0.26 when $N = 16, 32,$ and 64, respectively. We first measured the speedup with the ISP protocol, where the speedup was calculated with respect to the one-processor ISP execution. In the parallel implementations, the entities corresponding to a stage are always mapped to the same processor, with the entities assigned to processors using the "card-dealing" algorithm. As seen in Fig. 5, the speedup for ISP is relatively independent of N, which implies that the inherent parallelism available in each of the three configurations is not affected significantly by the number of links.[3]

3. The small reduction in speedup with $N = 64$ is due to the dramatic increase in the number of context switches for this configuration, as seen from Table 3.

Fig. 6, Fig. 7, and Fig. 8 show the efficiency of the three conservative protocols with $N = \{16, 32, 64\}$. As seen from the three graphs, the NULL and ANM protocols have similar efficiencies except for the sequential case, while the COND protocol performs quite differently. We examine the performance of each protocol in the following subsections.

5.2.1 Null Message Protocol

For the NULL protocol, the EIT value for an entity is calculated using EOT values from its incoming channels, i.e., from entities in its source-set. When the number of incoming channels to an entity is small, its EIT value can be advanced by a few null messages. Thus, the efficiency of the

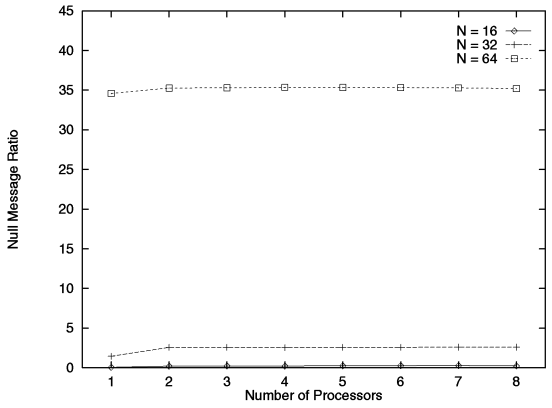

Fig. 9. CQNF: NMR in the NULL protocol.

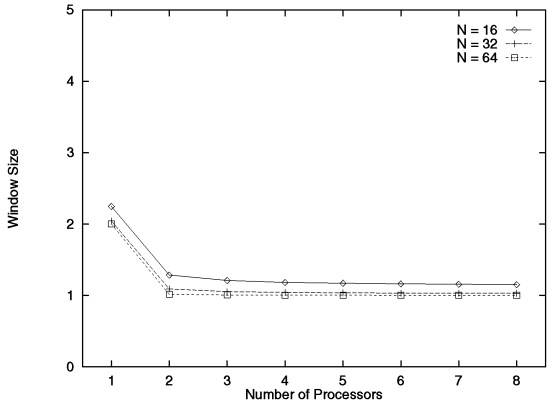

Fig. 10. CQNF: Window size in the COND protocol.

NULL protocol is relatively high in case of low connectivity, as shown in Fig. 6. However, with high connectivity, the protocol requires a large number of null messages and the performance degrades significantly. The null message ratio (NMR) is a commonly used performance metric for this protocol. It is defined as the ratio of the number of null messages to the number of regular messages processed in the model. As seen in Fig. 9, the NMR for $N = 16$ is quite low and increases substantially for $N = 64$ when the connectivity of the model becomes dense. As an entity must send null messages even to other entities on the same processor, NMR for each configuration does not change with the number of processors. However, the efficiency decreases as the number of processors is increased because null messages to remote processors are more expensive than the processor-local null messages.

5.2.2 Conditional Event Protocol

Unlike the other conservative protocols, the performance of the COND protocol is not affected by the communication topology of the model because the cost of the global ECOT computation is independent of the topology. Thus, in Fig. 6, Fig. 7, and Fig. 8, the efficiency of the COND protocol does not degrade; rather it improves as N goes from 32 to 64. This effect can be explained by looking at Table 3, which presents the number of context switches incurred in the one processor execution for each of the three model configurations with each protocol. With high connectivity, the likelihood that every entity has a "safe" message is low. Therefore, as seen in the table, even ISP needs to have more context switches for those models. The COND and GEL protocols, on the other hand, already have a high number of context switches for the model with low connectivity, thus the degradation caused by the entity scheduling scheme cannot be seen in the performance of the COND protocol. As a result, the performance of the COND protocol, relative to ISP, appears to improve for a model with high connectivity.

TABLE 3
Number of Context Switches in One Processor Executions
[$\times 1,000$]

	Regular Messages	ISP	GEL	NULL	COND	ANM
$N = 16$	1,177	33	1,266	95	1,127	271
$N = 32$	1,193	85	1,257	393	1,149	914
$N = 64$	1,229	259	1,254	4,571	1,171	1,182

The average window size is another commonly used metric for this protocol, which is defined as:

$$\text{Window size} = \frac{H}{\text{Number of synchronizations.}}$$

Fig. 10 shows the average window size for each of the three experiments with the COND protocol. As seen from the figure, the average window size in the parallel implementations is close to 1, which implies that the protocol requires one global computation for every simulation clock tick. The one-processor execution has a window size of 2. The smaller window size for the parallel implementation is caused by the need to *flush* messages that may be in transit while computing the global ECOT, as described in Section 4.4. However, the value of the window size (W) does not appear to have a strong correlation with the performance of the COND protocol: First, although W drops by 50 percent as we go from one to two processors, the efficiency does not drop as dramatically. Second, although the efficiency of the protocol continues to decrease as the number of processors (P) increases in Fig. 6, Fig. 7, and Fig. 8, Fig. 10 shows that W does not change dramatically for $P > 2$.

The performance degradation of this protocol with number of processors may be explained as follows: First, as each processor broadcasts its ECOT messages, each global ECOT computation requires $P(P-1)$ messages in a P processor execution ($P > 1$). Second, the idle time spent by an LP while waiting for the completion of each global ECOT computation also grows significantly as P increases. This can be seen from the fact that the COND protocol degrades its performance more than the other two as P increases in Fig. 6, Fig. 7, and Fig. 8.

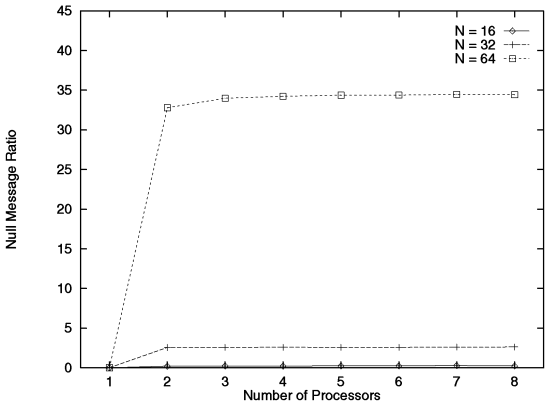

Fig. 11. CQNF: NMR in the AMN protocol.

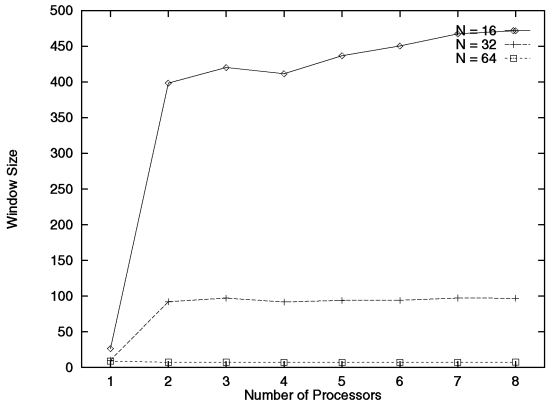

Fig. 12. CQNF: Window size in the ANM protocol.

5.2.3 ANM Protocol

The efficiency of the ANM protocol shown in Fig. 6, Fig. 7, and Fig. 8 is close to that of the NULL protocol except for the one processor execution. Fig. 11 and Fig. 12 show the NMR and the average window size, respectively. The ANM protocol has almost the same NMR values as the NULL protocol. The average window sizes vary widely for all the models although the window size becomes narrower as N increases. This occurs because of the following reasons:

- ECOT computation is initiated only when no entity on a processor has any messages to process. As N increases, since every processor must send one ECOT message for one global ECOT computation, the EIT updates by ECOT messages can easily defer unless every processor has few entities to schedule.
- Even if a global ECOT calculation is unhindered, it may have no effect if null messages have already advanced EIT values. In such cases, the number of null messages is not decreased by the global ECOT computations and the ANM protocol will have higher overheads than the NULL protocol.

Therefore, the efficiencies of the ANM protocol in Fig. 6, Fig. 7, and Fig. 8 are close to the ones of the NULL protocol except for the one processor executions. In the models with $N = \{16, 32\}$, ECOT messages are rarely sent out since null messages can efficiently advance the EIT value of each entity, and the situation where all the processors have no entity to schedule is unlikely. However, for $N = 64$, the performance of the ANM protocol is expected to be closer to that of the COND protocol since the EIT updates by ECOT messages are more efficient than the updates by null messages. As seen in Fig. 12, ECOT messages are in fact sent more frequently (the average window size is approximately 7) when $N = 64$. The inefficiency in this case arises because of the large number of null messages sent by each entity; hence, the duration when the processor would be idle in the COND protocol is filled up with the processing time of null messages. This can be seen by comparing Fig. 9 and Fig. 11 in which both protocols have almost the same NMR. Considering that the ANM protocol has more overhead caused by the ECOT computation, the reason that the efficiency is the same as the NULL protocol is that the ECOT messages actually improve the advancement of EIT a little, but the improvement makes up only for the overhead of ECOT computations.

In the experiments in this section, the global ECOT computations in the ANM protocol do not have sufficient impact to improve its performance. However, all the models in these experiments have good lookahead values, thus null messages can efficiently update EIT values of the entities. In the next section, we examine the cases where the simulation models have poor lookahead and compare the performance of the ANM and NULL protocols in this case.

5.3 Lookahead

5.3.1 No Precomputation of Service Times

It is well-known that the performance of conservative protocols depends largely on the lookahead value of the model. The lookahead for a stochastic server can be improved by precomputing service times [21], as shown in Fig. 2. In this section, we examine the effects of lookahead in the CQNF models by not precomputing the service times; instead, the lookahead is set to one time unit, which is the lower bound of the service time generated by the shifted exponential distribution. With this change, all entities have the lookahead value of one time unit, which is the worst possible value for this parameter. Fig. 13, Fig. 14, and Fig. 15 show the effect of this change where we plot D, the performance degradation that results from this change as a function of the number of processors. Note that the performance of ISP cannot be affected by the lookahead value because it does not use lookahead for the simulation run. Thus, the ISP performance is identical to the result shown in Fig. 5. The fractional performance degradation is calculated as follows:

$$D = \frac{\text{Execution time without precomputation of service times}}{\text{Execution time with precomputation of service times}} - 1.$$

Interestingly, the figures show that the performance degradation is negative in some cases for the NULL and

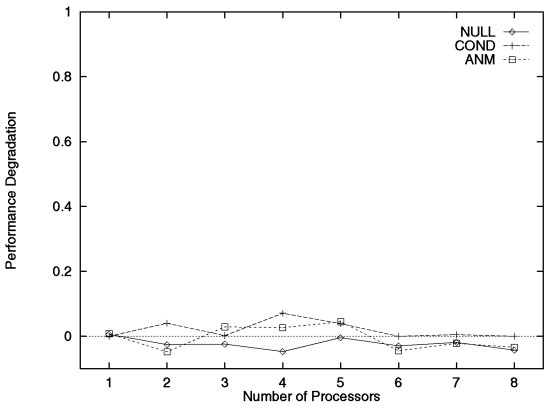

Fig. 13. CQNF: Performance with low lookahead ($N = 16$).

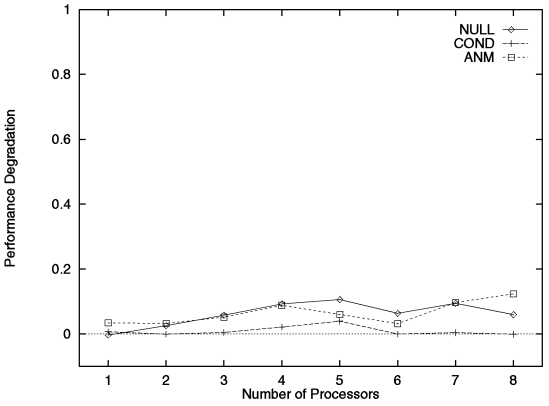

Fig. 14. CQNF: Performance with low lookahead ($N = 32$).

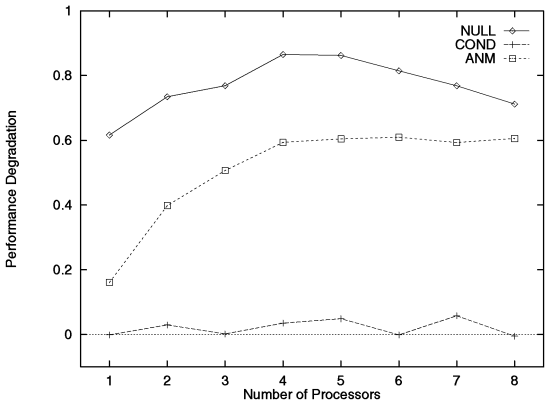

Fig. 15. CQNF: Performance with low lookahead ($N = 64$).

ANM protocols, which indicates that some executions actually become *faster* when lookahead is reduced. This counterintuitive behavior is attributed to the choice of the scheduling strategy discussed in Section 4.2; although this strategy reduces entity scheduling overheads of each processor, some entities may not be scheduled for a long period of physical time. When an entity is not scheduled, it does not send any null or regular messages to its *dest-set*, which may block the progress of other entities. If the blocked entities are on remote processors, this may increase the overall idle time of the execution. Note that the

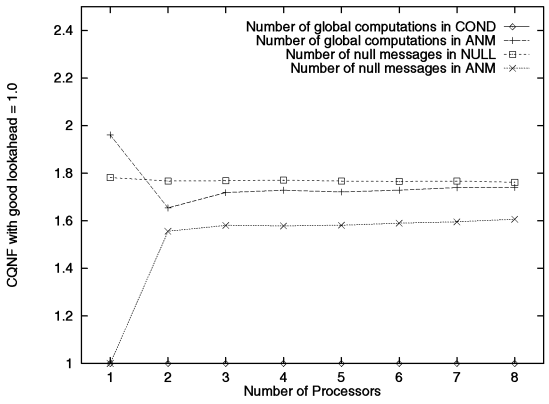

Fig. 16. CQNF: Increase of null messages and global computations with low lookahead ($N = 64$).

performance improvement with poorer lookahead occurs primarily in the configuration with $N = 16$, where the communication topology is not dense and the number of jobs available at each server is typically greater than 1, which may lead to long scheduling cycles.

The performance of the COND protocol does not degrade for all the models with poor lookahead because its window size is already small in the experiment with better lookahead. Note that it outperforms the other two protocols for $N = 64$ with better lookahead, as described in the previous section (Fig. 8); this indicates that the COND protocol has a significant advantage over the null message based protocols in the case where the simulation model has high connectivity or poor lookahead.

Another interesting observation is that the performance of the ANM protocol does not degrade as much as that of the NULL protocol in Fig. 15. This behavior may be understood from Fig. 16, which shows the relative increase in the number of null messages and global ECOT computations between the poor lookahead experiment considered in this section and the good lookahead scenario of Section 5.2. As seen in Fig. 16, for the model with poor lookahead, the ANM protocol suppresses the increase of null messages by increasing the number of global ECOT computations. This result shows the capability of the ANM protocol to implicitly switch the EIT computation base to ECOT messages when the EIT values cannot be calculated efficiently with null messages.

5.3.2 CQN with Priority Servers (CQNP)

In Section 5.3.1, we examined the impact of lookahead by explicitly reducing the lookahead value for each server entity. In this section, we vary the lookahead by using priority servers rather than FIFO servers in the tandem queues. We assume that each job can be in one of two priority classes: low or high, where a low priority job can be preempted by a high priority job. In this case, precomputation of service time cannot always improve the lookahead because, when a low priority job is in service, it may be interrupted at any time so the lookahead can at most be the

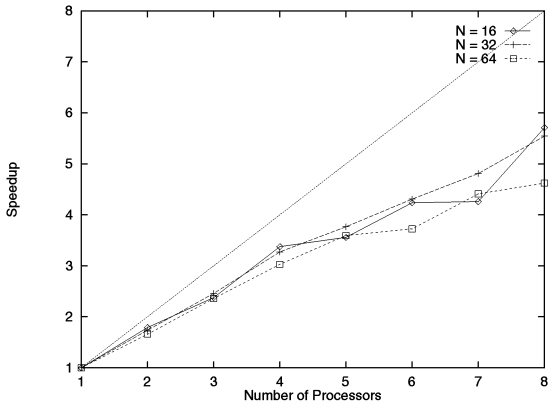

Fig. 17. CQNF: Speedup achieved by ISP.

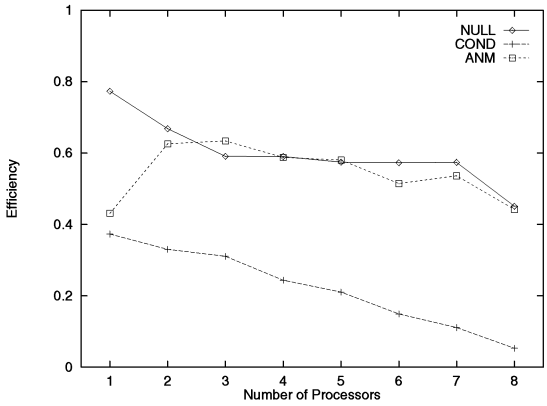

Fig. 18. CQNP: Efficiency of each protocol ($N = 16$).

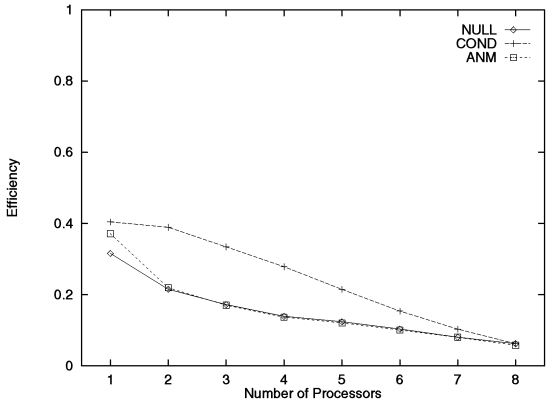

Fig. 19. CQNP: Efficiency of each protocol ($N = 32$).

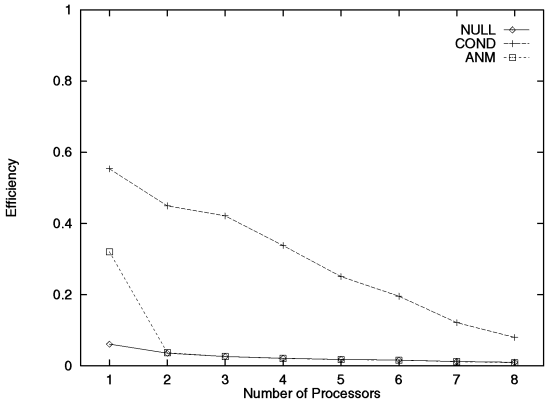

Fig. 20. CQNP: Efficiency of each protocol ($N = 64$).

remaining service time for the low priority job. Precomputed service times are useful only when the server is idle. The experiments described in this section investigate the impact of this variability in the lookahead of the priority server on the performance of each of the three protocols.

Fig. 17 shows the speedup achieved by ISP in the CQN models with priority servers (CQNP), where 25 percent of the jobs (256) are assumed to have a high priority. In comparison with Fig. 5, ISP achieves slightly better performance because the events in priority servers have slightly higher computation granularity than those in the FCFS servers. The impact of computation granularity on protocol performance is explored further in the next section. Since the ISP performance does not depend on the lookahead of the model, no performance degradation was expected or observed for this protocol.

Fig. 18, Fig. 19, and Fig. 20 show the efficiency of each protocol for $N = \{16, 32, 64\}$. The performance of the COND protocol with the priority servers is very similar to that with the FCFS servers (Fig 6, Fig. 7, and Fig. 8), while the performance of the other two protocols is considerably worse with the priority servers. Even though the connectivity is the same, the COND protocol outperforms the other two protocols for $N = 32$, which shows that the COND protocol is not only connectivity insensitive, but also less lookahead dependent than the null message based protocols.

5.4 Load Balancing

The CQN models used in the experiments so far have a homogeneous message flow. In this section, we examine the performance of a CQN model with asymmetric message flows. In the original CQN model, a switch can forward jobs to all switches with equal probability. In the modified model, the switch in Stage 1 still routes messages uniformly; however, all other switches route messages only to a subset of the stages: A job departing from switch i ($1 < i \leq N$) can be sent with equal probability only to one of the switches $1 \ldots j$ ($1 \leq j \leq i$). To ensure that this does not lead to instability in the network, the service time of the servers is also made nonuniform: S_i, the service time of servers in stage i, is set to $(i-1)*S_1$. Two different mappings were tried: In the first, the stages are assigned to a processor using a block mapping, whereas, in the second, they are assigned to the processors using the card dealing assignment strategy. In the block mapping, the model is partitioned among P processors, such that $P-1$ processors get $[N/P]$ stages, where $[N/P]$ is the integer part of N/P, and the Pth processor gets the rest. It should be clear that, in the first mapping, the workload is distributed nonuniformly, with the processors getting the lower numbered stages having higher workload.

Fig. 23 shows the maximum speedups predicted by ISP for the model with $N = 32$ and $H = 1,000,000$ with the two

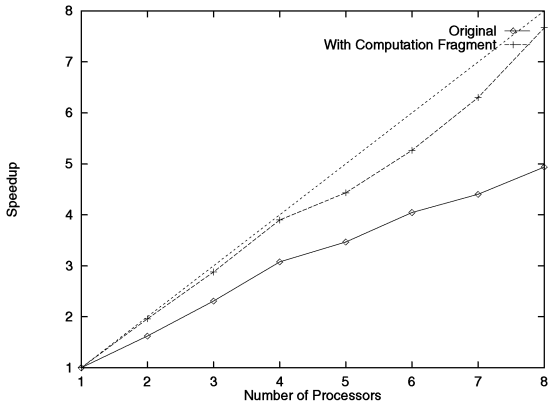

Fig. 21. CQNF: Speedup with synthetic computation fragment ($N = 32$).

partitioning schemes. As expected, the ISP performance with the block mapping is significantly worse than that for the CQN model with homogeneous message flows (Section 5.2, Fig. 5) because of the inherent lack of parallelism in this decomposition. However, the efficiencies of the conservative protocols (Fig. 24) are better than those obtained for the homogeneous mapping (Fig. 7), with the difference becoming larger as P increases. The better efficiency of the protocols for the model with the asymmetric message flow can be explained as follows: The processor with the heaviest workload needs fewer null messages in the execution using the NULL and ANM protocols because most of the EOT updates can be done with regular messages that are sent frequently in the model. The overhead for the COND protocol is also reduced because the idle time is less for processors with the heavier workload. The ISP performance for the model with the card-deal mapping is considerably better than that for the same model with the block mapping; and the potential speedups in Fig. 23 are very close to the ones in Fig. 5 in the configurations with less than four processors. With a few processors, the card-deal mapping evens out the imbalance of processor workloads; however, as more processors are used, the mapping scheme cannot resolve the workload problem when fewer entities are mapped on each processor.

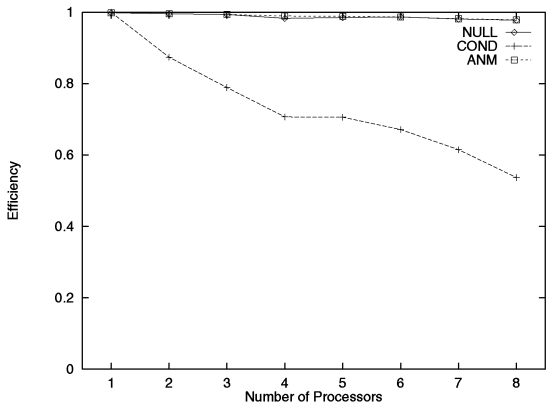

Fig. 22. CQNF: Efficiency of each protocol with synthetic computation fragment ($N = 32$).

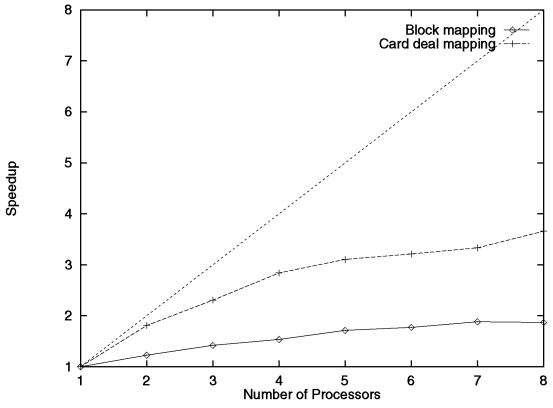

Fig. 23. Asymmetric CQNF: Speedup predicted by ISP.

The protocol efficiencies (Fig. 25) with the card-deal mapping display a similar trend being close to the ones shown in Fig. 7 for fewer processors and close to the ones shown in Fig. 24 when more processors are used.

5.5 Computation Granularity

One of the useful characteristics of the CQN models for evaluation of parallel simulation protocols is its fine computation granularity. Computation granularity is a protocol-independent factor, but it changes the breakdown of the costs required for different operations. For a model with high computation granularity, the percentage contribution of the protocol-dependent factors to the execution time is expected to be sufficiently small so as to produce relatively high efficiency for all the simulation protocols. We examine this by inserting a synthetic computation fragment containing 1,000,000 operations which is executed for every incoming "Job" message at each server. This increases the average computation granularity in the model to 38.69 milliseconds as compared to the 51.48 microseconds that had been used in all previous experiments. Fig. 21 shows the impact of the computation granularity on speedup using ISP for the CQNF model with $N = 32$. The total simulation time H is reduced to 1,000, as the sequential execution time for the model with the dummy computation

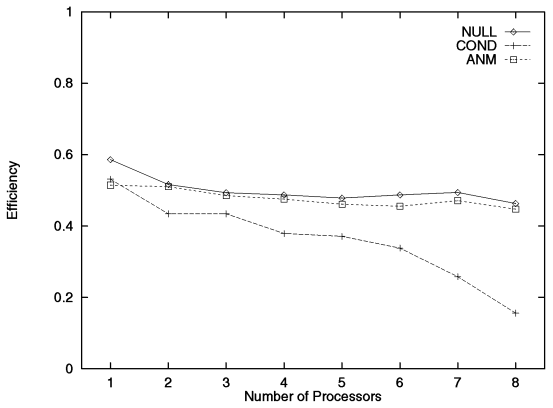

Fig. 24. Asymmetric CQNF: Protocol efficiency with block mapping.

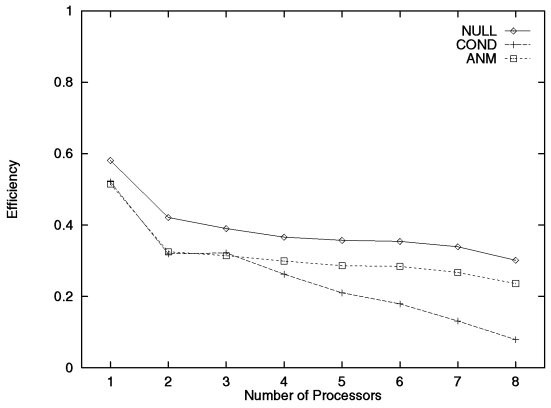

Fig. 25. Asymmetric CQNF: Protocol efficiency with card-dealing mapping.

fragment is 750 times longer than the original model. In the figure, the performance improvement with ISP is close to linear. The performance degrades a little when the number of processors is five, six, or seven because the decomposition of the model among the processors leads to an unbalanced workload, which is one of the *protocol-independent* factors in parallel simulation.

Fig. 22 shows the impact of increasing the computation granularity on the efficiency of the three conservative protocols. Clearly, the additional computation diminishes the protocol-dependent overheads in the NULL and the ANM protocols, and the performance of these two protocols is very close to that of ISP. The performance of the COND protocol, however, gets worse as the number of processors increases, and the efficiency is only 54 percent with eight processors. As described in Section 5.2.2, this degradation is due to the idle time that is spent by a processor while it is waiting for the completion of each global computation. This implies that, even if the COND protocol employs a very efficient method in global ECOT computation and the overhead for it becomes close to zero, the protocol can only achieve half the speedup of the ISP execution with eight processors because of the idle time that must be spent waiting for the global ECOT computation to complete. This result shows the inherent *synchronousness* of the COND protocol, although our implementation of the algorithm is asynchronous.

6 Related Work

Berry and Jefferson [1] introduced the critical path metric to derive theoretical bounds on the performance of distributed simulations. Among many simplifications, they assumed the existence of an unlimited number of processors to simplify the analysis and avoid overheads due to LP scheduling strategies. Recent work has led to a number of extensions to this metric. In particular, Lin [17] has presented parallelism analyzers that consider three event (process) scheduling schemes. The analyzers do not require tracing and use analytical formulations to compute estimated parallel execution time during the sequential execution of the model. There has also been some work showing that, in theory, an optimistic approach with lazy cancellation has the potential to outperform the lower bound on the parallel execution estimated by a critical path analysis (Jefferson and Reiher [14], Gunter [13]). Nicol [20] presents an analytical model to qualify the overhead due to various protocol-specific factors. For analytical tractability, the paper ignores some costs for protocol-independent operations and uses a simplified communication topology in the physical system. The preceding set of analytical techniques is useful for qualitative comparisons among models, but does not provide the tight upper bound on potential performance that can be derived using experimental tools like ISP.

In the area of experimental tools, Swope and Fujimoto [25] describe Osim, which is similar to ISP in its use of a simulation trace to identify precise event dependencies. However, unlike ISP, Osim uses analytical models to estimate a subset of the overhead costs of parallel simulations including the overhead for system calls and interprocessor communications. Because ISP includes all protocol-independent overheads that are actually present in a given mapping of a simulation model on a specific parallel architecture, it yields a tight lower bound on the execution time that can be used both as a basis for quantitative comparison among different simulation protocols as well as to identify the potential parallelism that can be exploited from a given decomposition of a simulation model.

Prior work on evaluating performance of discrete-event models with conservative protocols have used analytical and simulation models, as well as experimental studies. A number of previous experimental studies have evaluated the performance of conservative algorithms. Reed et al. [22] studied the performance of the deadlock avoidance algorithm and the deadlock detection and recovery algorithm on shared memory architecture and concluded that these algorithms did not yield substantial performance enhancements. Fujimoto [12] studied the null message deadlock avoidance algorithm [8] using queuing networks and synthetic benchmarks and showed that models with good lookahead can effectively exploit parallel execution. Wagner and Lazowska [26] analyzed the benchmark used in the study reported in [22] and showed that lack of inherent parallelism in the models partly accounted for the poor performance that was observed. Chandy and Sherman [9] reported excellent performance for a synchronous implementation of the conditional event algorithm using queuing networks that contained a very large number of jobs and had a relatively large computational granularity. The paper did not compare the performance of the conditional event protocol with that of others. Lubachevsky [18] describes how the synchronous algorithms work for scalable architectures with a limited number of interactions among processors using a notion of bounded lag. The conditional event algorithm studied in this paper corresponds to this bounded lag approach with no window. Blanchard et al. [7] have presented an algorithm that

combines the null message algorithm with bounded lag approach [18]. Although the implementation to combine two different protocols in the study is different from the accelerated null message algorithm in how to carry extra information (ECOT), it shares the same idea that an additional (synchronous) protocol compensates for the performance degradation of null message-based protocol in models where loops cause a flood of null messages.

This paper builds on the previous work by suggesting an approach based on the use of a common simulation language (Maisie) and a common reference point (ISP) for performance comparison of parallel simulation protocols. This approach was used to compare the relative performance of three diverse conservative algorithms as a function of significant model characteristics. Although previous studies have examined the relationship between certain model characteristics and overall impact on its performance with a given protocol, using ISP, we isolate the impact of changing model characteristics on the inherent parallelism in a model as well as on the overheads of a specific synchronization protocol.

7 Conclusions

An important goal of parallel simulation research is to facilitate its use by the discrete-event simulation community. Maisie is a simulation language that separates the simulation model from the specific algorithm (sequential or parallel) that is used to execute the model. Transparent sequential and optimistic implementations of Maisie have been developed and described previously [2]. This paper studied the performance of three diverse conservative algorithms that have been implemented in Maisie: a synchronous algorithm (conditional event), an asynchronous algorithm (null message), and a hybrid algorithm (ANM) that combines features from the preceding algorithms. Maisie models were developed for standard queuing network benchmarks, and various configurations of the model were executed using the three different algorithms. The benchmark was configured with a very low computational granularity in order to clearly understand the relationship between the other model characteristics and the specific overheads of a given simulation protocol. The results of the performance study may be summarized as follows:

- The Ideal Simulation Protocol (ISP) provides a suitable basis to compare the performance of conservative protocols as it separates the protocol-dependent and protocol-independent sources of overhead in a model. Unlike other existing metrics, like speedup and throughput, that only reflect the overall performance of parallel simulation, the separation of these factors precisely identifies the synchronization overhead in an implementation and allows even diverse protocols to be compared in a consistent manner. ISP also gives additional insight to the analyst as to whether the problem is due to inherent lack of parallelism in the model or due to the overheads in the specific synchronization protocol that is used for its execution. Other metrics, like NMR (Null Message Ratio) and the window size, are useful for the characterization of the protocol dependent overheads for the null message and conditional event protocols, respectively, but they cannot be used to compare diverse simulation protocols.
- The available parallelism in the model is unaffected by the degree of connectivity in the model (Fig. 5) or its lookahead properties. It decreases substantially with load imbalance (Fig. 23) and increases significantly with improvements in computational granularity (Fig. 21). The analysis clearly shows that an imbalance in the computational load leads to poor performance of the simulator simply because the inherent parallelism in the model decreases dramatically (Fig. 5 and Fig. 23) rather than due to an increase in the overhead of the protocols (Fig. 7, Fig. 24, and Fig. 25). In contrast, improvements in the computation granularity increase the inherent parallelism in the model and also decrease overheads for some protocols (Fig. 22) to yield overall improvements in performance.
- In general, for models with a low computational granularity, performance of the conservative protocols dropped significantly as the level of connectivity in the network was increased (Fig. 7 and Fig. 8), lookahead became poor (Fig. 14 and Fig. 15), or the load became unbalanced (Fig. 23 and Fig. 24). The impact of these factors differed among the protocols, as discussed next.
- Consistent with previous studies, this study found that the null message algorithm performed well for models with good lookahead properties and low connectivity. Our study showed that, in such scenarios, even models with very low computational granularity were able to exploit 70-80 percent of the available parallelism (Fig. 6). However, with a low computation granularity, even small increases in the connectivity in the model led to a dramatic reduction in its efficiency (Fig. 7 and Fig. 8), which could only be recouped by increasing the computation granularity (Fig. 22). Note that the ISP results showed that the potential parallelism in the model was only marginally affected by the connectivity (Fig. 5).
- Because of its insensitivity to the communication topology of the model, the conditional event protocol outperforms the null message based protocols when the connectivity is higher (Fig. 8) or when the lookahead is poor (Fig. 15, Fig. 19, and Fig. 20). Although the synchronous nature of the algorithm degrades its performance in parallel executions with a high number of processors (Fig. 22), the performance of this protocol was the most stable among the protocols compared in this study. This algorithm also has the additional advantage of not requiring positive lookahead values; unlike the null message-based

techniques, this algorithm may also be used to simulate models with zero lookahead cycles.

- The performance of the ANM protocol is between those of the null message and the conditional event protocols for one processor execution (Fig. 4). The performance in parallel executions, however, is very close to that of the null message protocol (Fig. 6, Fig. 7, Fig. 8, Fig. 18, Fig. 19, and Fig. 20) since small EIT advancements by the null messages defer the global ECOT computation (Fig. 11 and Fig. 12). Thus, it performs worse than the conditional event protocol when the simulation model has high connectivity (Fig. 8) or very poor lookahead (Fig. 19 and Fig. 20). However, compared to the null message protocol in such cases, the ANM protocol prevents an explosion in the number of null messages with frequent global ECOT computations (Fig. 16) and performs better than the null message protocol (Fig. 15). As this algorithm also inherits the characteristic of the conditional event algorithm that can simulate the models with zero lookahead cycles, it is best suited for models whose lookahead properties are unknown.
- All the protocols studied in this paper failed to exploit the available parallelism for models with dense connectivity. In general, the need for frequent communication among LPs in the model increases synchronization overheads; hence, this problem may be inherent to all the synchronization protocols. However, the impact of the model characteristic differs in each protocol; for instance, the performance of conditional event protocol almost remains the same as N changes (Fig. 6, Fig. 7, Fig. 8, Fig. 18, Fig. 19, and Fig. 20). Further investigation with other protocols, including optimistic ones, is needed to identify the most appropriate protocol for such models.

Note that while the evaluation of the algorithm performed with synthetic workloads provides a fair amount of insight into the ability of the algorithm to handle a variety of cases as demonstrated in this paper, it is certainly desirable to evaluate the algorithm under varying real practical workload conditions. Such measurements will not only validate the synthetic workloads but also provide an assessment of the algorithm's value in practice. A number of studies have already been completed to evaluate the performance of applications like network protocols [4], VLSI circuit models [10], and wireless systems [23] using the simulation algorithms and language studied in this paper.

In the future, we intend to expand on the performance study reported in this paper to include other algorithms, including optimistic and adaptive protocols, as well as to evaluate a larger class of architectures, including distributed memory machines like the IBM SP and scalable shared memory architectures like the SGI Origin 2000.

Acknowledgments

The authors wish to thank Professor Iyer and the three reviewers for their valuable comments and suggestions that resulted in significant improvements to the paper. We also acknowledge the assistance of past and present team members in the Parallel Computing Laboratory for their role in developing the Maisie software. Maisie may be obtained by anonymous ftp from http://pcl.cs.ucla.edu/projects/maisie. This research was partially supported by the U.S. Department of Defense/Advanced Research Projects Agency ARPA/CSTO under contract DABT-63-94-C-0080 and the U.S. Defense Advanced Research Projects Agency DARPA/ITO under contract DAAB-07-97-C-D321.

References

[1] O. Berry and D. Jefferson, "Critical Path Analysis of Distributed Simulation," *Proc. 1985 SCS Multiconf. Distributed Simulation,* pp. 57–60, Jan. 1985.

[2] R. Bagrodia, K.M. Chandy, and W. Liao, "A Unifying Framework for Distributed Simulations," *ACM Trans. Modeling and Computer Simulation,* vol. 1, no. 4, pp. 348–385, Oct. 1991.

[3] R. Bagrodia and W. Liao, "Maisie: A Language for Design of Efficient Discrete-Event Simulations," *IEEE Trans. Software Eng.,* vol. 20, no. 4, pp. 225–238, Apr. 1994.

[4] R. Bagrodia, Y. Chen, M. Gerla, B. Kwan, J. Martin, P. Palnati, and S. Walton, "Parallel Simulation of a High-Speed Wormhole Routing Network," *Proc. 1996 Workshop Parallel and Distributed Simulation,* pp. 47–56, May 1996.

[5] R. Bagrodia, "Language Support for Parallel Discrete-Event Simulations," *Proc. Winter Simulation Conf.,* pp. 1,324–1,331, Dec. 1994.

[6] S. Bellenot, "Global Virtual Time Algorithms," *Proc. Multiconf. Distributed Simulation,* vol. 22, no. 1, pp. 122-127, Jan. 1990.

[7] T.D. Blanchard, T.W. Lake, and S.J. Turner, "Cooperative Acceleration: Robust Conservative Distributed Discrete Event Simulation," *Proc. 1994 Workshop Parallel and Distributed Simulation,* pp. 58–64, July 1994.

[8] K.M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Comm. ACM,* vol. 24, no. 11, pp. 198–206, Aug. 1981.

[9] K.M. Chandy and R. Sherman, "The Conditional Event Approach to Distributed Simulation," *Proc. 1989 Simulation Multiconf.: Distributed Simulation,* vol. 21, no. 2, pp. 93–99, Mar. 1989.

[10] Y. Chen and R. Bagrodia, "Shared Memory Implementation of a Parallel Switch-Level Circuit Simulator," *Proc. 1998 Workshop Parallel and Distributed Simulation,* pp. 134–141, May 1998.

[11] R. Fujimoto, "Parallel Discrete Event Simulation," *Comm. ACM,* vol. 33, no. 10, pp. 30–53, Oct. 1990.

[12] R.M. Fujimoto, "Performance Measurements of Distributed Simulation Strategies," *Trans. Soc. Computer Simulation,* vol. 6, no. 2, pp. 89–132, Apr. 1989.

[13] M.A. Gunter, "Understanding Supercritical Speedup," *Proc. 1994 Workshop Parallel and Distributed Simulation,* pp. 81–86, July 1994.

[14] D. Jefferson and P. Reiher, "Supercritical Speedup," *Proc. 24th Ann. Simulation Symp.,* pp. 159–168, Apr. 1991.

[15] V. Jha and R. Bagrodia, "Transparent Implementation of Conservative Algorithms in Parallel Simulation Languages," *Proc. 1993 Winter Simulation Conf.,* pp. 677–686, Dec. 1993.

[16] V. Jha, and R. Bagrodia, "A Performance Evaluation Methodology for Parallel Simulation Protocols," *Proc. 10th Workshop Parallel and Distributed Simulation,* pp. 180–185, May 1996.

[17] Y.B. Lin, "Parallelism Analyzers for Parallel Discrete Event Simulation," *ACM Trans. Modeling and Computer Simulation,* vol. 2, no. 3, pp. 239–264, July 1992.

[18] B.D. Lubachevsky, "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks," *Comm. ACM,* vol. 32, no. 1, pp. 111–131, Jan. 1989.

[19] J. Misra, "Distributed Discrete-Event Simulation," *ACM Computing Surveys,* vol. 18, no. 1, pp. 39–65, Mar. 1986.

[20] D.M. Nicol, "The Cost of Conservative Synchronization in Parallel Discrete-Event Simulations," *J. ACM,* vol. 40, no. 2, pp. 304–333, Apr. 1993

[21] D.M. Nicol, "Parallel Discrete Event Simulation of FCFS Stochastic Queuing Networks," *Parallel Programming: Experience with Applications, Languages and Systems, ACM SIGPLAN,* pp. 124–137, July 1988.

[22] D.A. Reed, A.D. Malony, and B.D. McCredie, "Parallel Discrete Event Simulation Using Shared Memory," *IEEE Trans. Software Eng.,* vol. 14, no. 4, pp. 541–553, Apr. 1988.

[23] J. Short, R. Bagrodia, and L. Kleinrock, "Mobile Wireless Network System Simulation," *Proc. First Ann. Int'l Conf. Mobile Computing and Networking,* pp. 195–209, Nov. 1995.
[24] W.K. Su and C.L. Seitz, "Variants of the Chandy-Misra-Bryant Distributed Simulation Algorithm," *Proc. 1989 Simulation Multiconf.: Distributed Simulation,* vol. 21, no. 2, pp. 38–43, Mar. 1989.
[25] S.M. Swope and R.M. Fujimoto, "Optimal Performance of Distributed Simulation Programs," *Proc. Winter Simulation Conf.,* pp. 612–617, Dec. 1987
[26] D. Wagner and E. Lazowska, "Parallel Simulation of Queuing Networks: Limitations and Potentials," Technical Report No. 88-09-05, Computer Science Dept., Univ. of Washington, Sept. 1988.

Rajive Bagrodia received a BTech degree in electrical engineering from the Indian Institute of Technology, Bombay, in 1981. He obtained the MA and PhD degrees in computer science from the University of Texas at Austin in 1983 and 1987, respectively. He is a professor of computer science at the University of California at Los Angeles. Professor Bagrodia's research interests include parallel simulation, parallel languages and compilers, and nomadic computing. He has published more than 100 research papers on the preceding topics. The research has been funded by a variety of government and industrial sponsors including the US National Science Foundation, US Office of Naval Research, US Advanced Research Projects Agency (DARPA), Rome Laboratory, and Rockwell International. He is an associate editor of the *ACM Transactions on Modeling and Computer Systems* (*TOMACS*). He was selected as a 1991 Presidential Young Investigator by the US National Science Foundation. He is a senior member of the IEEE and the IEEE Computer Society.

Mineo Takai received the BS, MS, and PhD degrees, all in electrical engineering, from Waseda University, Tokyo, Japan, in 1992, 1994, and 1997, respectively. Since 1997, he has been with the Computer Science Department at the University of California, Los Angeles, where he is currently a principal development engineer. Dr. Takai's current research involves design and development of high performance simulation tools for large-scale networks. His research interests include parallel and distributed computing, parallel discrete event simulation, and mobile computing. He is a member of the IEEE and the IEEE Computer Society.