Performance Evaluation of Conservative Algorithms in Parallel Simulation Languages

Rajive L. Bagrodia

Mineo Takai

Vikas Jha

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 Maisie, an algorithm-independent simulation language. The algorithms include the null message algorithm, the conditional event algorithm, and a new algorithm called Accelerated Null Message algorithm that combines the preceding alternatives. The experiments presented in this paper study the performance of the algorithms as a function of model connectivity, computation granularity, and lookahead properties – model characteristics that have the most significant impact on the performance of conservative protocols. A novel contribution of this paper is the introduction of the Ideal Simulation Protocol (or ISP) as a common reference point to compare the efficiency of the algorithms.

Keywords: Discrete-event Simulation, Parallel and Distributed Simulation, Conservative Algorithms, Parallel Simulation Languages, Lookahead, algortihmic 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 [9, 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[4]. 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 the null message algorithm[6], the conditional event[7] algorithm, and a new conservative 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. We define the efficiency of a protocol using the notion of the Ideal Simulation Protocol or ISP, as introduced in this paper.

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 including both hardware and software overheads, and the overheads of the parallel synchronization algorithm. When a parallel model fails to yield expected performance benefits, the analyst has few tools with which to ascertain the underlying cause. For instance, it is difficult to determine whether the problem is due to inherent lack of parallelism in the model or due to large overheads in the implementation of the synchronization protocol. The Ideal Simulation Protocol offers a partial solution to this problem. ISP allows an analyst to experimentally identify the maximum parallelism that 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 the percentage degradation in performance that is 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. Previous work has relied on theoretical critical path analyses to compute lower bounds on model execution times. These bounds can be very approximate because they ignore all overheads, including architectural and system overheads over which the parallel simulation algorithm has no control.

The remainder of the paper is organized as follows: the next section 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 $\textit{dest-set}_p$ and $\textit{source-set}_p$ 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 teh 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 re-computations. In this paper, we focus on the conservative algorithms.

In order to simplify the description of the algorithms, we define the following terms. Each term is defined for an LP_p at physical time r. We assume that the communication channels are FIFO.

- **Earliest Input Time** ($EIT_p(r)$): Lower bound on the (logical) timestamp of any message that LP_p may *receive* in the interval (r, ∞).

- **Earliest Output Time** ($EOT_p(r)$): Lower bound on the timestamp of any message that LP_p may *send* in the interval (r, ∞).

- **Earliest Conditional Output Time** ($ECOT_p(r)$): Lower bound on the timestamp of any message that LP_p may *send* in the interval (r, ∞) *assuming that* LP_p *will not receive any more messages in that interval.*

- **Lookahead** ($la_p(r)$): 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. Figure 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

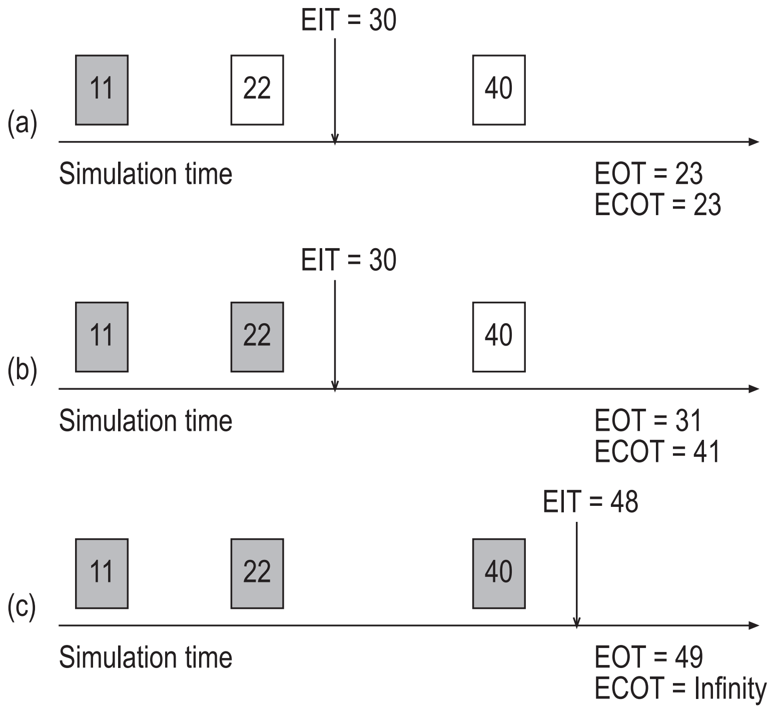

Figure 1: The computation of EOT and ECOT: The LP is modeling a server with a minimum service time of one time unit. Shaded messages have been processed by the LP.

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 (a), since $T_{next} = 22 < EIT = 30$, both EIT and $ECOT$ are equal to T_{next} plus the minimum service time. In (b), 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 (c), $T_{next} = \infty$ because there are no unprocessed messages. Therefore, $ECOT$ is equal to ∞.

Various conservative algorithms differ in how they compute the value of EIT for each LP in the system. By definition, in a conservative algorithm, at physical time r, LP_p can only process messages with timestamp $\leq EIT_p(r)$. 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. 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 piggy-backed with regular messages or they may be sent only when an LP is blocked, rather than whenever there is a change in its EOT.

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)[13].

2.2 Conditional Event Algorithm

In the conditional event algorithm [7], 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-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 [5] 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. 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 presence of good lookahead with the ability of conditional event algorithm to execute even without lookahead – a scenario in which null message algorithm alone will deadlock. In models with poor lookahead where it may take many

[1] Transitive closure of *source-set* in many applications is almost the same as the set of *all* LPs in the system.

rounds of null messages to sufficiently advance the EIT of an LP, ANM could directly compute the earliest global event considerably faster. Message piggy-backing is used extensively to reduce the number of synchronization messages, and the global ECOT computation at a node is initiated only when the node is otherwise blocked.

3 Ideal Simulation Protocol

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. 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 refer 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 teh 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[6] or conditional event messages[7], 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). In the past, critical path analyses have been used to prove theoretical lower bounds and related properties[12, 1] of a parallel simulation model. However, such analyses do not include the cost of many *protocol-independent* factors in the computation of the critical path time. Excluding these overheads, which are not contributed by the simulation protocol, means that the computed bound is typically a very loose lower bound on the execution time of the model, and is of relatively little practical utility to the simulationist either in improving the performance of a given model or in measuringthe efficiency of a given protocol.

We introduce the notion of an Ideal Simulation Protocol (ISP), that is used to experimentally compute a tight lower bound on the execution time of a simulation model on a given architecture. Although ISP is based on the notion of critical path, it computes the parallel execution time by actually *executing* the model on the parallel architecture. The resulting lower bound predicted by ISP is realistic because it includes overheads due to protocol-independent factors that must be incurred by any simulation protocol that is used for its execution, and assumes that the synchronization overheads are zero.

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. As no synchronization protocol is used to execute the model using ISP, the measured execution time will not include any protocol-specific overheads. However, unlike the critical path analyses, the ISP-predicted bound will include the cost of all the *protocol-independent* factors that are required in the parallel execution of a model. Given this lower bound, it is easy to measure the efficiency of a protocol as described in the next section.

Besides serving as a reference point for computing the efficiency of a given protocol, a representative pre-simulation 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 applictaion. 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. In addition, Maisie provides constructs to specify the dynamic communication topology of a model and the lookahead properties of each entity. These constructs were used to investigate the impact of different communictaion topologies and lookahead patterns on the performance of each of the conservative algorithms described earlier.

Figure 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 (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). Subsequently it 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, 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 dynamically add or remove elements from these sets: **add_source**(), **del_source**(), **add_dest**(), and **del_dest**(). In Figure 2, the server entity specifies its connectivity to the other entities using the **add_source**() and **add_dest**() functions (line 8–9).

Lookahead is another important determinant of performance for conservative algorithms. Maisie provides a pre-defined 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 then may be feasible otherwise. For instance, when the server is idle, it is possible

```
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  }
```

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

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.

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 sub-optimal 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 have been discussed in [17]. The experiments found that 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. At the start of a new phase i, the jth processor computes its E_{ij} and M_{ij}. E_{ij} is the ECOT value for the phase i, and M_{ij} is the smallest timestamp that is sent by the processor in the phase $i-1$. M_{ij} accounts for the messages in transit; in the worst case none of the messages sent by the processor in the phase $i-1$ may have been received and accounted for in E_i of other processors. Thus, the global ECOT for the phase i is calculated as the following:

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

where n is the number of processors. Messages sent during phases $i-2$ and earlier do not need to be considered 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 $i-2$.

During the phase i, every processor sends an ECOT message which contains the minimum of its E_i and M_i, and can compute the global ECOT to be the minimum of ECOT messages from all the 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 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 at the beginning of the simulation in order to exclude the time of reading the trace file during execution. To reduce the matching time, the trace is stored simply as a sequence of the unique numeric identifiers that are assigned to each message by the runtime system to ensure FIFO operation of the input queue. Thus the only 'synchronization' overhead in executing a simulation model with ISP is the time required to execute a bounded number of numeric comparison operations for each incoming message. This cost is clearly negligible when compared with other overheads of parallel simulation and the resulting time is an excellent lower bound on the exection time for a parallel simulation.

5 Experiments and Results

The programs used for the experiments were written in Maisie and contain additional directives for parallel simulation, i.e. (a) explicit assignment of Maisie entities to processors, (b) code to create the source and destination sets for each entity, and (c) specification of lookaheads. All experimental measurements were taken on a SPARCserver 1000 with 8 SuperSPARC processors, and 512M of shared main memory.

We selected the Closed Queuing Networks (CQN), a widely used benchmark, to evaluate parallel simulation algorithms. This benchmark was used because it is easily reproducible and it allows the communication topology and lookahead, the two primary determinants of performance of the conservative protocols, to be modified in a controlled manner. 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 the results for the CQNP experiments.

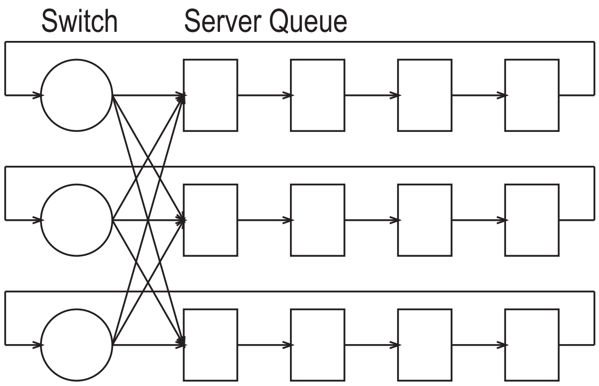

Figure 3: A Closed Queuing Network($N = 3$, $Q = 4$)

- 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).
- 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 entities corresponding to a switch and its associated tandem queue servers are always mapped to the same processor. The shifted-exponential distribution is chosen as the service time in our experiments so that the minimum lookahead for every entity is non-zero, thus preventing a potential deadlock situation with the null message algorithm. The topology of the network, for $N = 3$ and $Q = 4$, is shown in Figure 3. The performance of parallel algorithms is typically presented by computing speedup, where

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

However, as we will describe in the next section, the sequential execution time of a model using the standard Global Event List (GEL) algorithm[13] does not provide 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 become saturated if the model does not have enough parallelism. In this paper, we suggest computing efficiency using ISP which does provide a tight lower bound, and then compare the conservative protocols with respect to their efficiency relative to the execution time with ISP. The efficiency of a protocol S on architecture A is defined as follows:

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

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) with comparing the execution time of each of the five protocols: GEL(global event list), NULL(null message), COND(conditional event), ANM, and ISP on a single processor. This configuration is instructive because it identifies the unique set of 'sequential' overheads incurred by each protocol on one processor. In subsequent sections, we investigate the impact of modifications in various model characteristics like model connectivity (section 5.2), lookahead properties (section 5.3), and computation granularity (section 5.4).

5.1 Simulation of a CQNF Model on One Processor

Figure 4 shows the execution times for a CQNF model with $N = 32$, $Q = 3$, $J = 1024$, $S = 10$, and $H = 100,000$ on one processor of 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. This makes the protocol-specific overheads associated with an event relatively large, allowing us to clearly compare these overheads for the various protocols, which is the primary purpose of the study.

In Figure 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 4.5, 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 1 shows the total number of context switches for each protocol for this configuration. The number of context switches for ISP is approximately 15 times

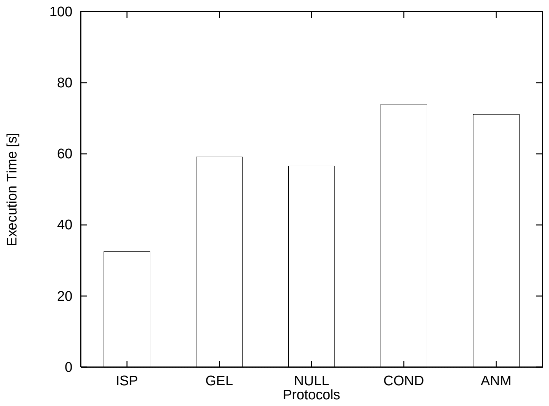

Figure 4: Execution times of a CQNF model on one processor

Table 1: 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 $[\times 1,000]$.

less than that required by the GEL algorithm. Considering that the total messages processed during the execution was about 1,200,000 messages, the GEL protocol does almost one context switch per message[2]. ISP and conservative protocols also use a more efficient data structure to order the entities than the GEL algorithm. Whereas GEL must use an ordered data structure like the splay tree to sort entities in the order of the earliest timestamps of messages in their input queues, the other protocols use a simple unordered list because the entities are scheduled on the basis of safe messages. 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.

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 is 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

[2] Since the Maisie runtime system has a main thread which does not correspond to any entity, the number of context switches can be larger than the number of messages.

has three times the number of context switches than the NULL protocol. This is because the COND protocol has to switch out all the entities on every global ECOT computation, which globally updates the EIT value of each entity, while the NULL protocol 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 1 time unit, thus the window size between two global ECOT computations is small. As shown in Table 1, the COND protocol had approximately 49,000 global computations, which averages to one global computation for every advance of 2 units in simulation time.

Even though the NULL protocol uses few context switches, its execution time is significantly larger than ISP because of the large number of null messages (1732,000 from Table 1) that must be processed and the resulting updates to EIT. Although the ANM protocol incurs overhead due to both null messages and global ECOT computations, both the number of null messages and global ECOT computations are significantly less than those for the NULL and COND protocols. As a result, its performance lies somewhere between that of the NULL and COND protocols.

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 serevrs were kept constant among the three configurations, by keeping $N(Q+1) = 128$. The other parameters for the model were set to $J = 1024$, $S = 10$, and $H = 100,000$. The total number of entities ($N(Q+1)$) is kept constant among the different configurations to also keep the size of the model constant. 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 total number of links is 368, 1120, and 4160 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. As seen in Figure 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]

Figure 6 to 8 show the efficiency of the three conservative protocols with $N = \{16, 32, 64\}$. As

[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 2.

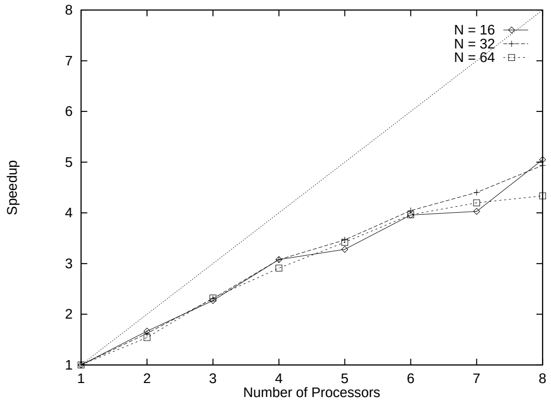

Figure 5: CQNF: Speedup achieved by ISP

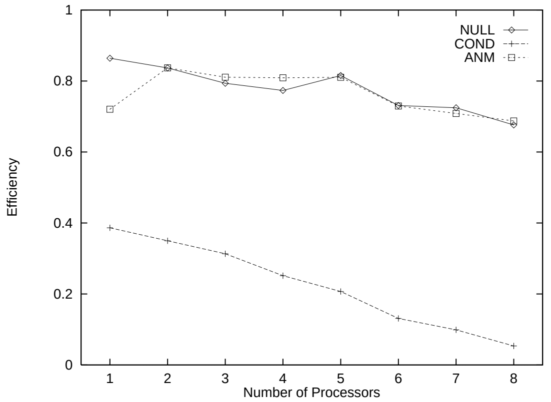

Figure 6: CQNF: Efficiency of each protocol ($N = 16$)

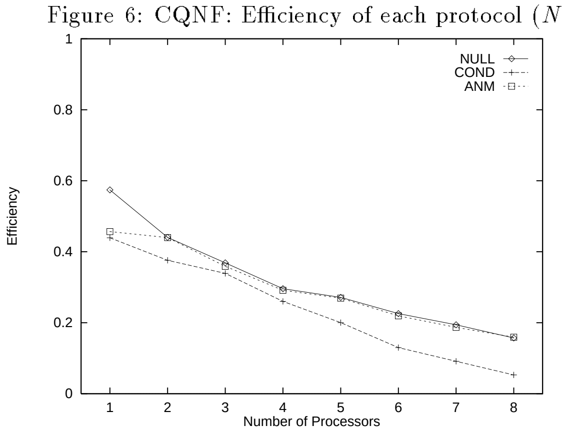

Figure 7: CQNF: Efficiency of each protocol ($N = 32$)

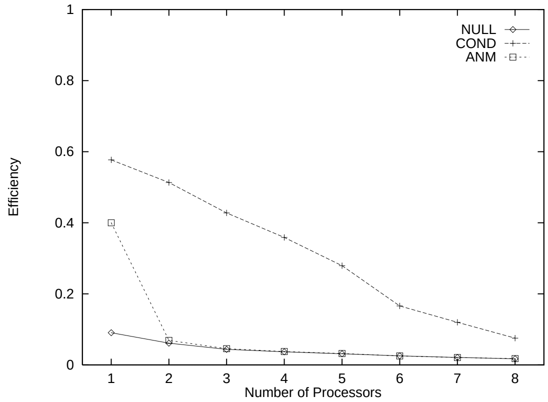

Figure 8: CQNF: Efficiency of each protocol ($N = 64$)

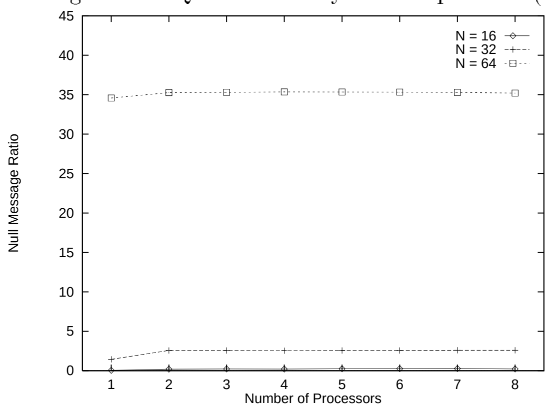

Figure 9: CQNF: NMR in the NULL protocol

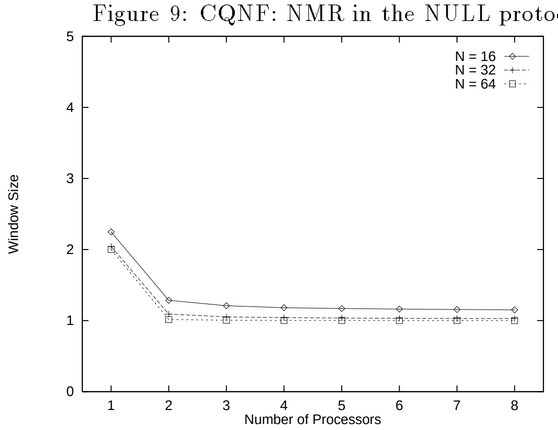

Figure 10: CQNF: Window size in the COND protocol

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 NULL protocol is relatively high in case of low connectivity, as shown in Figure 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 Figure 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 consrvative protocols, the performance of the COND protocol is not affected by the communictaion topology of the model because the cost of the global ECOT computation is independent of the topology. Thus, in Figures 6 to 8, the efficiency of the COND protocol does not degrade; rather it improves as N goes from 32 to 64. The reason for this is that relative to the COND protocol, the performance of ISP degrades more causing the relative efficiency of the COND protocol to improve! This effect can be explained by looking at Table 2, which presents the number of context switches incurred in the one processor execution for each of the three model configurations with each protocol. As seen in the table, this measure increases dramatically for ISP as N increases from 32 to 64, while the COND protocol already has one context switch per message with $N = 16$ and this does not change significantly as N increases.

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

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

Figure 10 shows the average window size for each of the three experiments with the COND protocol.

Table 2: 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

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% as we go from 1 to 2 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 Figures 6 – 8, Figure 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.

5.2.3 ANM Protocol

The efficiency of the ANM protocol shown in Figure 6 to 8 is close to that of the NULL protocol except for the one processor execution. Figure 11 and 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 Figure 6 to 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 Figure 12, when 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 Figure 9 and 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 does 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[14]. 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 the entities have the lookahead value of one time unit, which is the worst lookahead value of our experiments. Figures 13 to 15 show the effect of this change where we plot p, 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 Figure 5. The fractional performance degradation is calculated as follows:

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

Interestingly, the figures show that the performance degradation is negative in some cases, which indicates that some executions actually become *faster* when lookahead is reduced. For the NULL and ANM protocols, this is attributed to the choice of the scheduling strategy discussed in Section 4.2; although this strategy reduces context switching overheads, it may also cause *starvation* of other entities. An entity that is scheduled for a long period blocks the progress of other entities on the same processor, which may also block other entities on remote processors and thus increase the overall idle time. Note that the 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.

For the COND protocol, the duration of the window size determines how long each entity can be scheduled before a context switch. A longer window implies fewer ECOT computations which may also lead to increased blocking of other entities. Thus, in some cases, poorer lookahead may force frequent ECOT messages and thus improve performance. As this effect is not related to the communication topology, the relative performance of the COND protocol with and without lookahead is almost the same for all three values of N (Figures 13 to 15).

The performance of the NULL and ANM protocols degrade as the connectivity of the model becomes dense, while the performance of the COND protocol does not change significantly as discussed above. In particular, for $N = 64$, the performance of the COND protocol does not degrade, although it already outperforms the other two protocols in the experiment with better lookahead (Figure 8). This indicates that the COND protocol has 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 Figure 15. This behavior may be understood from Figure 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 Figure 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.

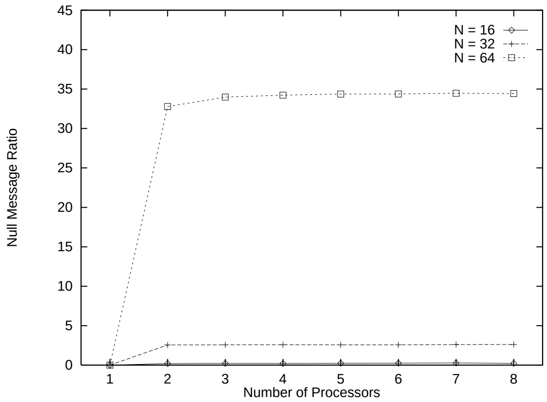

Figure 11: CQNF: NMR in the ANM protocol

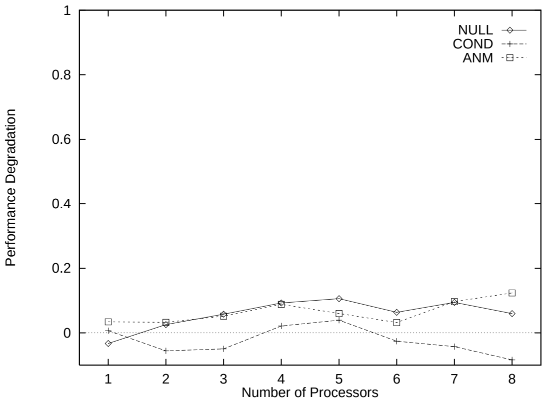

Figure 14: CQNF: Performance with low lookahead ($N = 32$)

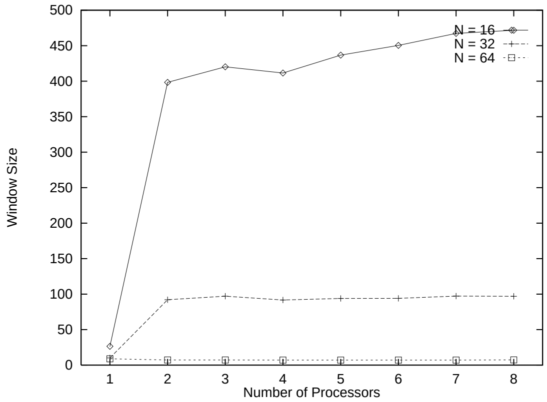

Figure 12: CQNF: Window size in the ANM protocol

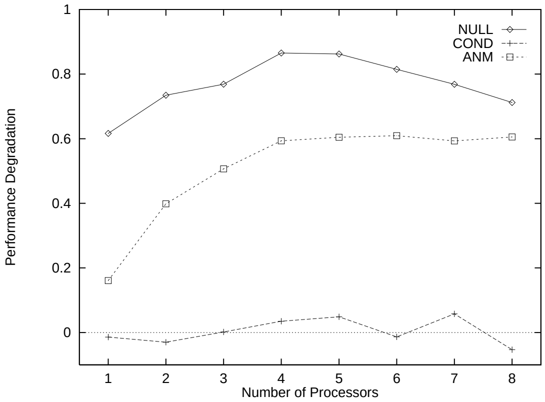

Figure 15: CQNF: Performance with low lookahead ($N = 64$)

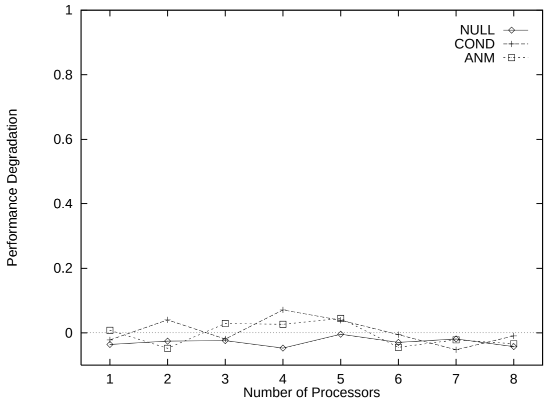

Figure 13: CQNF: Performance with low lookahead ($N = 16$)

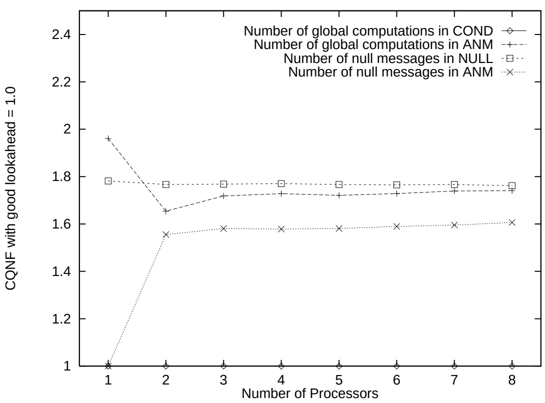

Figure 16: CQNF: Increase of null messages and global computations with low lookahead ($N = 64$)

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 imrove 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 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.

Figure 17 shows the speedup achieved by ISP in the CQN models with priority servers (CQNP), where 25% of the jobs (256) are assumed to have a high priority. In comparison with Figure 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.

Figures 18 to 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 (Figures 6 to 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 Computation Granularity

One of the good characteristics of the CQN models for the 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 will 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 message at each server. Figure 21 shows the impact of the computation granularity on speedup using ISP for the CQNF model with $N = 32$. The

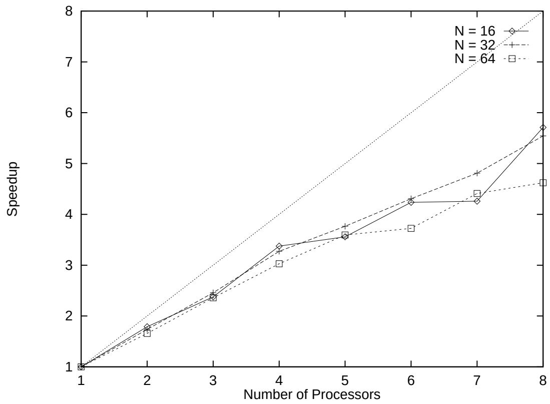

Figure 17: CQNP: Speedup achieved by ISP

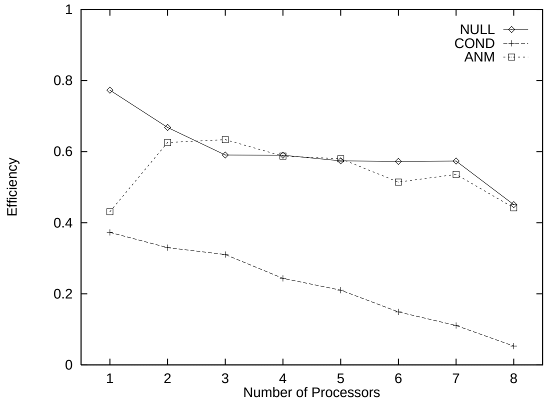

Figure 18: CQNP: Efficiency of each protocol ($N = 16$)

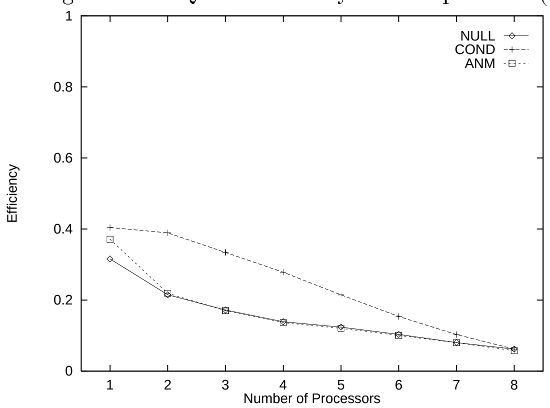

Figure 19: CQNP: Efficiency of each protocol ($N = 32$)

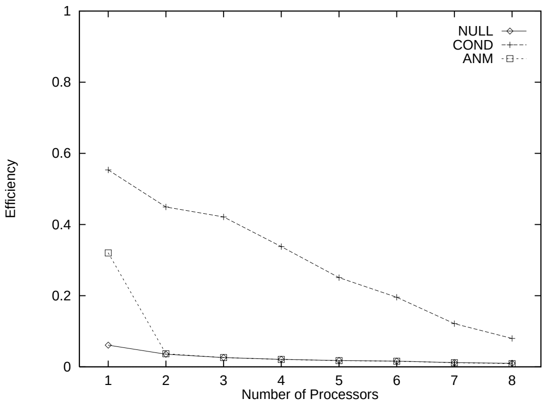

Figure 20: CQNP: Efficiency of each protocol ($N = 64$)

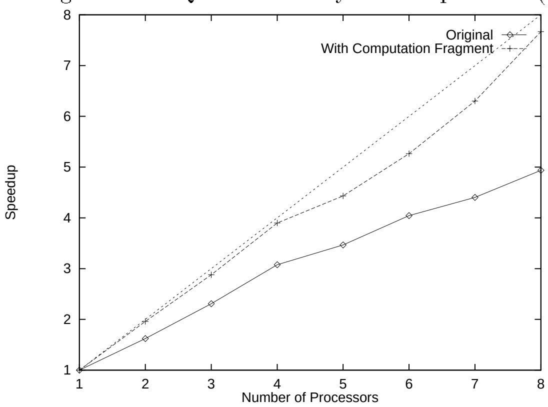

Figure 21: CQNF: Speedup with synthetic computation fragment ($N = 32$)

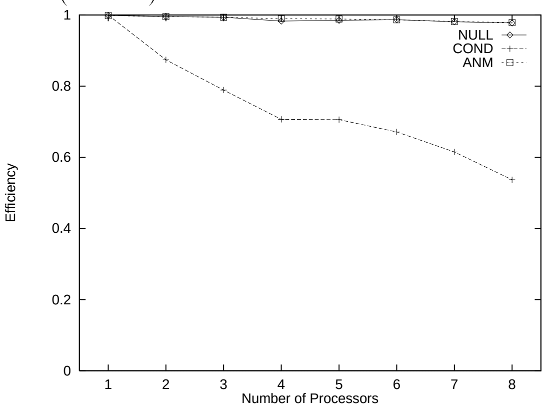

Figure 22: CQNF: Efficiency of each protocol with synthetic computation fragment ($N = 32$)

simulation horizon H is reduced to 1,000 as the sequential execution time for the model with the dummy computation fragment is 1,200 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 5, 6 or 7 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. The improvement in speedup with larger computation granularity can be attributed to an improvement in the ratio of the event processing time to the other protocol-independent factors such as communication latency and context switch times.

Figure 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% with 8 processors. As described in 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 8 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

Prior work on evaluating performance of discrete-event models with conservative protocols have used analytical and simulation models, as well as experimental studies.

Performance of the null message deadlock avoidance algorithm [6] using queuing networks and synthetic benchmarks has been studied by Fujimoto [10]. Reed et al [16] have studied the performance of the deadlock avoidance algorithm and the deadlock detection and recovery algorithm on shared memory architecture. Chandy and Sherman [7] describe the conditional event algorithm and study its performance using queuing networks. Their implementation of the conditional event algorithm is synchronous (*i.e.* all LPs carry out local computations followed by a global computation), and the performance studies carried out in the paper assume a network with very high number of jobs. The paper does not compare the performance of the conditional event protocol with others. Nicol[15] describes the overheads of a synchronous algorithm similar to the conditional event algorithm studied in this paper. A mathematical model is constructed to qualify the overhead due to various protocol-specific factors. However, for analytical tractability, the paper ignores some costs for protocol-independent operations or simplification of the communication topology of the physical system. Thus, the results are useful for qualitative comparisons but do not provide the tight upper bound on potential performance that can be derived using ISP.

The effect of lookahead on the performance of conservative protocols was studied by Fujimoto [10]. Nicol [14] introduced the idea of precomputing the service time to improve the lookahead. Cota and Sargent [8] have described the use of graphical representation of a process in automatically computing its lookahead. These performance studies are carried out with simulation models specific to their experiments.

The performance study presented in this paper differs in two important respects: first, we developed and used a new metric – efficiency with respect to the Ideal Simulation protocol, which allows the protocol-specific overheads to be separated cleanly from other overheads that are not directly contributed by a simulation protocol. Second, the implementation of each of the algorithms was separated from the model which allows the performance comparison to be more consistent than if the algorithm is implemented directly in the simulation model.

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 a variety of conservative algorithms that have been implemented in Maisie. The three algorithms that were studied include the null message algorithm, the conditional event algorithm, and a new algorithm called the accelerated null message algorithm that combines the preceding approaches. Maisie models were developed for standard queuing network benchmarks. Various configurations of the model were executed using the three different algorithms. 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 clearly separates the protocol dependent and independent factors that affect the performance of a given synchronization protocol. It gives a realistic lower bound on the execution time of a simulation model for a given partitioning and architecture. Existing metrics like speedup and throughput indicate *what* the performance is but do not provide additional insight into *how* it could be improved. 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 among the diverse set of simulation protocols. ISP, on the other hand, can be used by an analyst to compare the protocol-dependent overheads even between conservative and optimistic protocols.

- The null message algorithm exploited good parallelism in models with dense connectivity. However, the performance of this algorithm is very sensitive to the communication topology and lookahead characteristics of the model, and thus is not appropriate for models which have high connectivity or low lookahead values.

- Because of its insensitivity to the communication topology of the model, the conditional event protocol outperforms the of null message based protocols when the LPs are highly coupled. Also, it performs very well for models with poor lookahead. Although the algorithmic *synchronousness* degrades its performance in parallel executions with a high number of processors, the performance of this protocol is very stable throughout our experiments. With faster global ECOT computations, it may achieve a better performance. Since the conditional event algorithm also has a good characteristic of not requiring positive lookahead values; this algorithm may also be used to simulate models which have 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, as expected. The performance in parallel executions, however, is very close to that of the null message protocol since small EIT advancements by the null messages defer the global ECOT computation. Thus, it performs worse than the conditional event protocol when the simulation model has high connectivity or very poor lookahead. 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, and performs better than the null message protocol. This shows that the protocol has the capability for adaptively and implicitly switching its execution mode from the null message based synchronization to the conditional event based one for better performance. Since 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 properties are unknown.

Future work includes the use of ISP for performance evaluation of various algorithms such as optimistic algorithms and synchronous algorithms together with the algorithms examined in this paper, using additional applications from different disciplines.

Acknowledgments

The authors wish to thank Professor Iyer and the three reviewers for their valuable comments and suggestions that resulted in significnat improvements to the presentation. 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 annonymous ftp from http://may.cs.ucla.edu/projects/maisie.

References

[1] Kumar A. and R. Shorey. Stability of event synchronisation in distributed discrete event simulation. In *1994 Workshop on Parallel and Distributed Simulation*, Edinburgh, July 1994.

[2] R. Bagrodia, K.M. Chandy, and W. Liao. A unifying framework for distributed simulations. *ACM Trans on Modeling and Computer Simulation*, October 1991.

[3] R. Bagrodia and W. Liao. Maisie: A language for design of efficient discrete-event simulations. *IEEE Transactions on Software Engineering*, April 1994.

[4] R.L. Bagrodia. Language support for parallel discrete-event simulations. In *1994 Winter Simulation Conference*, Orlando, Florida, December 1994.

[5] S. Bellenot. Global virtual time algorithms. In *Multiconference on Distributed Simulation*, January 1990.

[6] K.M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(11):198–206, August 1981.

[7] K.M. Chandy and R. Sherman. The conditional event approach to distributed simulation. In *Distributed Simulation Conference*, Miami, 1989.

[8] B.A. Cota and R.G. Sargent. A framework for automatic lookahead computation in conservative distributed simulation. In *Multiconference on Distributed Simulation*, January 1990.

[9] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[10] R. M. Fujimoto. Performance measurements of distributed simulation strategies. Technical Report Tech. Rep. UUCS-87-026a, University of Utah, Salt Lake City, 1987.

[11] R.M. Fujimoto and D.M. Nicol. State of the art in parallel simulation. In *Winter Simulation Conference*, December 1992.

[12] Yi-Bing Lin. *Understanding the Limits of Optimistic and Conservative Parallel Simulation*. PhD thesis, University of Washington, Seattle, August 1990.

[13] J. Misra. Distributed discrete-event simulation. *Computing Surveys*, 18(1), March 1986.

[14] D.M. Nicol. Parallel discrete event simulation of fcfs stochastic queueing networks. In *Parallel Programming: Experience with Applications, Languages and Systems*, pages 124–137. ACM SIGPLAN, July 1988.

[15] D.M. Nicol. The cost of conservative synchronization in parallel discrete event simulations. *Journal of the ACM*, 40(2):304–333, April 1993.

[16] D.A. Reed, A.D. Malony, and B.D. McCredie. Parallel discrete event simulation: A shared memory approach. In *Proceedings of the 1987 ACM SIGMETRICS Conference*, pages 36–39, May 1987.

[17] Wen-king Su and C.L. Seitz. Variants of the chandy-misra-bryant distributed simulation algorithm. In *1989 Simulation Multiconference: Distributed Simulation*, Miami, Florida, March 1989.

- Affiliation of Authors: Rajive Bagrodia and Mineo Takai are with Computer Science Department, University of California, Los Angeles, CA 90095. Vikas Jha is with Inktomi Corporation, CA 94403. Email: <rajive/mineo@cs.ucla.edu, jha@inktomi.com>.

- This research was partially supported by the U.S. Department of Justice/Federal Bureau of Investigation/Advanced Research Projects Agency, ARPA/CSTO, under contract J-FBI-93-112 and the U.S. Department of Defense/Advanced Research Projects Agency ARPA/CSTO, under contract DABT-63-94-C-0080.

List of Footnotes

1. Transitive closure of source-set in many applications is almost the same as the set of *all* LPs in the system.

2. Since the Maisie runtime system has a main thread which does not correspond to any entity, the number of context switches can be larger than the number of messages.

List of Figures

List of Tables