Exploiting Temporal Uncertainty in Parallel and Distributed Simulations[1]

Richard M. Fujimoto
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

Abstract

Most work to date in parallel and distributed discrete event simulation is based on assigning precise time stamps to events, and time stamp order event processing. An alternative approach is examined where modelers use time intervals rather than precise time stamps to specify uncertainty as to when events occur. Partial orderings called approximate time (AT) and approximate time causal (ATC) order are proposed and synchronization algorithms developed that exploit these specifications to yield more efficient execution on parallel and distributed computers. Performance measurements of the AT-ordering mechanism on a cluster of workstations demonstrate as much as twenty-fold performance improvement compared to time stamp ordering with negligible impact on the results computed by the simulation. The context for much of this work is federated simulation systems that provided the initial motivation for this work. These results demonstrate that exploiting temporal uncertainty inherent in the simulation can lead to efficient parallel execution despite zero lookahead using conservative synchronization techniques, a long-standing problem in the parallel discrete event simulation field.

1 Introduction

Federated simulation systems represent perhaps the most widespread application of parallel and distributed simulation technology to date. Fueled by the substantial cost savings realized through model reuse and the observation that no one system can meet the requirements of all simulations in the U.S. Department of Defense, the High Level Architecture (HLA) effort is perhaps the most well known exploitation of this technology. The origins of the HLA can be traced back to early work in Simnet [1] and the ensuing Distributed Interactive Simulation (DIS) standards [2] focused on federated distributed simulation systems for training applications. Work in the Aggregate Level Simulation Protocol [3] extended this concept to analytic war gaming applications.

Federated simulations also offer the potential for solving an important problem in the parallel discrete event simulation community, namely, the cost and effort in parallelizing

[1] Supported by a DARPA contract under the Advanced Simulation Technology Thrust (ASTT) program.

a sequential simulation program. Rather than partitioning a large simulation program, one can federate smaller configurations of sub-models. For example, a simulation of a large telecommunication network can be realized by federating separate, stand alone sequential simulations of sub-networks. This approach was demonstrated in developing a parallel version of the CSIM package, for example [4].

Many federated simulations require time management mechanisms to ensure before and after relationships are correctly modeled by the simulation. The traditional approach to time management used by the parallel simulation community, and adopted by the HLA, is to assign a precise time stamp to each event, and ensure that events are processed in time stamp order. A synchronization protocol is needed to achieve the latter property. Synchronization protocols have been widely studied, and are often categorized as being either conservative, or optimistic. See [5] for an in-depth discussion of these protocols.

Unfortunately, existing conservative and optimistic synchronization mechanisms have severe liabilities when applied to federated simulation systems. It is well known that conservative simulations perform poorly when the model has little or no lookahead [6]. It is also well known that modifying the model to exploit lookahead is in general, time consuming, difficult, and often leads to models that are hard to maintain because modest changes to the model such as adding preemptive or state dependent behaviors may destroy the model's lookahead properties. Unfortunately, many applications, e.g., military wargaming simulations, require zero lookahead, and the size and complexity of these models often preclude modifying the model to exploit lookahead.

Although optimistic synchronization provides a potential solution to the lookahead problem, optimistic execution introduces other major obstacles. Most notably, a major re-engineering effort must be undertaken to modify the model to include a rollback capability. Minimally, the state of the model must be checkpointed using copy or incremental state saving techniques. While this process can sometimes be at least partially automated, modifying an existing model to include state saving is in general a major task because the changes must usually be made throughout the entire model, rather

than within a small number of modules. Further, optimistic execution requires many additional modifications to the simulation software. The existing I/O, error, and exception handling mechanisms must be reworked to allow for optimistic execution. Additional mechanisms may be needed to deal with execution errors that cannot occur in a conservative execution, but could occur (and be rolled back) with optimistic processing. Dynamic memory allocation mechanisms must also be modified to prevent memory leaks when rollbacks occur. While there are known technical solutions to most of these problems, a substantial amount of effort, and expense, is again required to rework an existing simulation to enable it to execute with an optimistic synchronization protocol.

Another problem not addressed by existing time management mechanisms concerns reuse of simulations across traditionally different application domains. For example, in the HLA one can envision reuse of event driven war game simulations in training environments to populate a virtual environment intended for human-in-the-loop tank simulations. Even though each event driven simulation may be able to keep up with real time, synchronization overheads and blocking imposed to ensure time stamp ordered event processing may prevent the simulation from achieving real-time performance in a federated simulation environment. Further, training simulations typically impose more relaxed event ordering requirements (e.g., DIS simulators are usually *not* required to process events in time stamp order) than event driven simulations. One would like to relax the ordering constraints of the event driven simulations in order to achieve faster execution, but it is not clear how this can be accomplished other than completely turning off the event ordering mechanisms, which could cause the simulator to fail. A mechanism that allows the modeler to carefully tailor ordering constraints according to model and usage requirements would facilitate reuse across domains.

In short, existing conservative and optimistic synchronization mechanisms fall short when applied to federated simulations for many applications, particularly those requiring zero lookahead. Further, synchronization is a very mature area of research in the parallel discrete event simulation community. Thus, it appears unlikely that new algorithms for realizing time stamp order event processing will solve these problems. This suggests a

new line of attack using semantics *other than* time stamp order processing of events may be required to provide a practical time management mechanism for federated simulations.

This paper describes one such approach to time management that exploits temporal uncertainty in the simulation model. The objective of this approach is to achieve more efficient executions, despite zero lookahead, and without resorting to optimistic synchronization techniques. Specifically, two new message ordering services, called approximate time (AT) and approximate time causal (ATC) order are proposed to replace time stamp order event processing. The next section reviews related work in relaxing event ordering constraints. The semantics of the AT and ATC ordering mechanisms are then described, as well as distributed algorithms to realize them. Experimental results are then presented to evaluate the impact of relaxing ordering constraints in zero lookahead simulations both in terms of execution speed and accuracy of the simulation results in a cluster computing environment. Areas for future work are then described.

The context for this work is federated simulation systems. However, the ideas presented here are equally applicable to non-federated parallel and distributed simulations. In particular, the ordering mechanisms presented here can be used to speed up conservative parallel simulations containing little lookahead. The approach described here might also be used to enhance the performance of optimistic synchronization mechanisms by reducing the amount of rollback.

2 Related Work

A limited amount of work in the parallel discrete event simulation community has examined the issue of relaxing ordering constraints. For example, early versions of the Moving Time Window protocol allowed out-of-order event processing [7]. More recently, ordering constraints were relaxed in modeling wireless networks, and found not to have a significant impact on the simulation results [8]. Such work has essentially focused on allowing out-of-order processing of events when that is convenient or expedient to the synchronization protocol. These techniques offer limited control by the

modeler concerning acceptable event orderings; we believe detailed modeler control is critical for an approach to achieve widespread acceptance. Further, these approaches do not provide clear semantics concerning what orderings are possible in a way that is independent of the simulation application and synchronization protocol.

Time intervals and use of fuzzy logic have been studied as a means for specifying temporal or other types of uncertainty in simulation applications, e.g., see [9, 10]. This work is does not address issues concerning model execution on parallel and distributed computing platforms, however, the central focus of the work described here. Prior work in time intervals is complementary to the work presented here in that it addresses issues concerning the assignment and manipulation of intervals within the model, while here, the focus is on concurrent execution of the model once the intervals have been defined.

Finally, other work in the distributed computing community has focused non-time stamp based ordering mechanisms. In particular, causal ordering has been extensively studied for general, distributed computing applications e.g., see [11, 12]. However, for the most part this work does not consider requirements of simulation application. For example, simulations often use time stamps to order actions, e.g., air traffic might be arranged according to pre-defined flight schedules. Causal ordering, by itself, does not provide a means to guarantee, for instance, proper interleaving of such time sequenced events in different processors. Causal order can be exploited to eliminate certain temporal anomalies, however, and is used in the ATC ordering mechanism proposed here.

3 Exploiting Temporal Uncertainty

Existing synchronization protocols studied in the parallel discrete event simulation community assume each event is assigned a time stamp value, and events must be processed in time stamp order. This is a natural approach that originates from classical (sequential) discrete event simulation methods. Though widely used, assigning a specific time stamp to an event is somewhat presumptuous because in reality, there is almost always a certain amount of uncertainty as to exactly when an event occurs. One can immediately think of many examples illustrating this fact. For example:

- The time a moving vehicle comes into view depends on how fast the vehicle is traveling, which depends on uncertain road conditions and other factors that cannot be known with complete certainty.
- The time the next packet is emitted from a traffic source in a telecommunications network simulation often depends on the operations and reaction time of human users that are being modeled, which cannot be known with complete certainty.
- The think time required by human operators in issuing orders in a military simulation cannot be determined with complete certainty.

It is clear that temporal uncertainty is ubiquitous in simulation modeling, stemming from the fact that virtual any simulation is only an approximation of the real world. Thus, there is no *a priori* reason to believe there is only one correct ordering of events that is acceptable during the execution of the simulation model. Yet, time stamp ordering implicitly makes precisely this claim, and parallel discrete event simulation protocols have had to adhere to this assumption since their inception. It is conceivable that relaxing time stamp ordering can yield much more efficient parallel and distributed simulation synchronization protocols without compromising the validity of the simulation results that are produced. Further, because different simulations have different requirements with respect to accuracy and realism, one can envision tailoring the order relaxation mechanism to the intended use of the simulation, e.g., greater relaxation of ordering constraints when used in a virtual environment setting as opposed to an analytic application. The approach described here allows for such relaxation, without requiring recompilation of the simulation program.

Two ordering mechanisms are proposed to relax time stamp order: approximate time order, and approximate time causal order. Both rely on exploiting temporal uncertainty inherent in the simulation model in order to improve the efficiency of the synchronization mechanism.

4 Approximate Time Order (AT Order)

Both AT and AT-causal ordering use simulation time intervals rather than precise time values. Specifically, each event X is assigned a time interval [E(X), L(X)] where E(X) ≤

L(X). E(X) denotes the earliest point in simulation time that the event may occur, and L(X) the latest. Time intervals are assigned to the event by the simulation model, typically based on uncertainty regarding when the event occurs. We assume the parallel/distributed simulation consists of a collection of logical processes (LPs) that communicate exclusively by exchanging messages. In a federated simulation system, each federate may include one or more LPs.

4.1 Event Ordering

AT order defines a partial ordering of events based on the *is-before* ordering relation (represented ~>). Consider two events X and Y with time intervals [E(X),L(X)] and [E(Y),L(Y)] respectively. The is-before relationship is defined as follows:

- *Non-overlapping time intervals:* if L(X) < E(Y), then X ~> Y (read X *is before* Y or X *precedes* Y). For example, in Figure 1(b) A ~> E.
- *Overlapping time intervals:* if the two time intervals have at least one point in common, then no ordering relationship exists between these events. In this case, the events are said to be *concurrent*, represented X || Y. In Figure 1(b) A || B.

The ~> relationship is transitive, i.e., if X~>Y, and Y~>Z, then X~>Z. The || relationship is *not* transitive. In Figure 1(b) D||E and E||F, but D and F are *not* concurrent.

Approximate time order is identical to time stamp order if E(X) is equal to L(X) for each event. If L(X) is equal to infinity for each event, then no ordering constraints apply because all events are concurrent. Thus, both conventional parallel discrete event simulation protocols and the DIS protocol that does not impose ordering constraints on events are special cases of AT-order.

A key benefit of time intervals is they can be used to increase the concurrency in the simulation. Figure 1(a) shows a snapshot of a simulation using time stamp order. The events shown in this figure have different time stamps, so they cannot, a priori, be considered to be concurrent. Concurrent processing of these events is achieved in existing synchronization mechanisms by relying on lookahead constraints, or by

providing the ability to undo the processing of an event. On the other hand, if the events are assigned time intervals as shown in Figure 1(b), events A, B, C and D are concurrent and can be processed simultaneously on different processors.

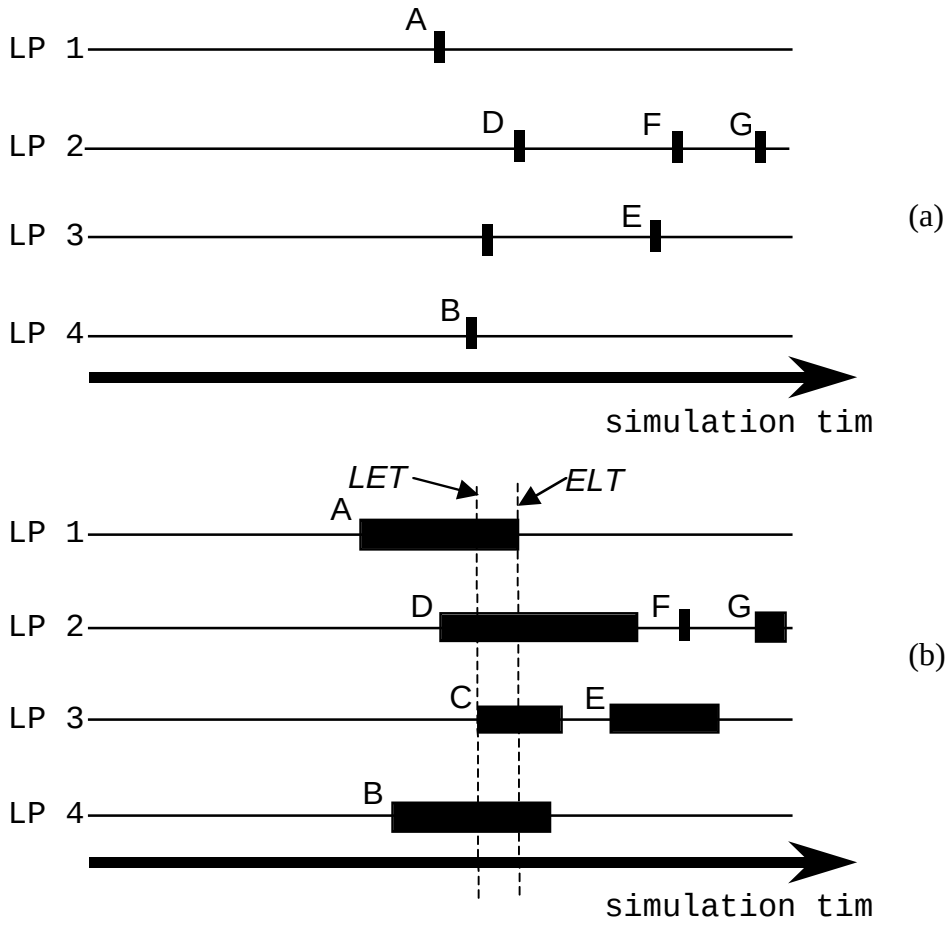

Figure 1. Example illustrating time intervals and AT-order. (a) events using time stamp order. (b) events using time intervals.

It is sometimes convenient to assign a precise time stamp to an event specified using a time interval. In particular, when AT-order is used to federate existing time stamp ordered simulations, such simulators expect events to be assigned precise time stamps. If each event is assigned a precise time stamp that lies within the time stamp interval for that event, the resulting event ordering will always be consistent with AT-order, i.e., if X ~> Y, then TS(X) < TS(Y) where TS(E) is the time stamp assigned to event E.

4.2 AT Simulations

A discrete event simulation based on AT-order utilizes the following principles:

- Each LP processes events in AT-order. For any two events X and Y residing within a single LP, if X~>Y, then X must be processed before Y. If X || Y, then X and Y may be processed in any order.
- A simulation time clock is defined. The current simulation time of each LP is defined as an interval $[C_E,C_L]$, indicating the LP's current time is greater than or equal to C_E, but no larger than C_L. In a conservative execution, the E-time of the clock never decreases. This ensures that if C_i and C_{i+1} are successive clock values, either $C_i \parallel C_{i+1}$, or $C_i \sim> C_{i+1}$. An event X whose time stamp interval overlaps with $C=[C_E,C_L]$ is said to be concurrent with C (X||C).
- If a federate's current simulation time is $C=[C_E,C_L]$ and that federate schedules a new event X, then $E(X) \geq C_E$. This ensures either X || C or C ~> X for any new event X scheduled by the federate during time interval C. This constraint is referred to as the *AT-scheduling constraint.*

Lookahead constraints, well known in conservative synchronization algorithms, can be used in AT-order simulations. Specifically, if the lookahead of an LP is L, then each event X generated by a federate at time $[C_E,C_L]$ adheres to the constraint $E(X) \geq C_E + L$. Large lookahead is not a prerequisite to achieving high degrees of concurrency in AT-order simulations, but can be exploited when it exists. The focus of this work is on zero lookahead simulations, however, for completeness lookahead will be included in the discussions that follow where appropriate.

4.3 AT-Order Synchronization Algorithms

The synchronization algorithm must ensure events are processed in AT-order and no event is delivered to an LP in its past, i.e., no event X is delivered such that X~>C where C is the LP's clock. Two synchronization algorithms are described next. The first is a simple, synchronous algorithm that does not attempt to exploit lookahead. The second is a more sophisticated, asynchronous algorithm that can exploit knowledge concerning lookahead and minimum time stamp intervals, when available.

4.3.1 A Simple, Synchronous Algorithm

The first algorithm, shown in Figure 2, executes a loop that simply identifies the earliest set of concurrent events S_E in the simulation, and then processes them. The algorithm performs a barrier synchronization at the end of each iteration to ensure events in the current iteration have been processed before advancing to the next iteration. The barrier must ensure there are no transient messages in the network before releasing any processor; this can be accomplished using message counters to keep track of the number of messages sent and received. In Figure 1(b) S_E in the first iteration includes four events A, B, C, and D. After processing these events, the second iteration would process events E and F, and the third event G, assuming no new events are created during these iterations.

An important aspect of the algorithm is identifying the earliest set of concurrent events. This can be accomplished by computing the global minimum among the L-times of all unprocessed events. This minimum value is referred to as the *earliest L-time* or ELT. This value is shown in Figure 1(b), and it can be seen the events in S_E, A, B, C, and D, are the four events that overlap with ELT. The following lemma verifies that the set of events whose time interval overlap with ELT are concurrent events, and no events in the set of pending events precedes these events in approximate time order.

```
Clock [C_E,C_L] = [0,0]
while (at least one unprocessed event remains)
  ELT = min (L(X)) for all unprocessed events X
  S_E = {all events Y: E(Y) ≤ ELT ≤ L(Y)}
  LET = max (E(X)) for all X ∈ S_E
  Clock [C_E,C_L] = [LET, ELT]
  process events in S_E
  barrier
end-while
```

Figure 2. Algorithm 1: a simple AT-order simulation algorithm.

Lemma 1. Consider a non-empty set of events S with each event assigned a time stamp interval. Let ELT be the minimum L-time among the events in S. Let S_E be the set of events {X: E(X) ≤ ELT ≤ L(X)}, i.e., the set of events that include ELT in their time stamp interval. Then for any event X ∈ S_E, there is no event Z∈ S, such that Z ~> X. Further for any two events X, Y ∈ S_E, X || Y.

Proof. There must be at least one event in S_E because S is non-empty, and the event with L-time equal to ELT must be in S_E. Let X be any event in S_E. Then E(X) ≤ ELT ≤ L(X). Let Z be some other event in S. If Z ~> X, then L(Z) < E(X) ≤ ELT, but this would contradict the fact that ELT is the minimum L-time among all events in S. Thus no such event Z can exist.

For the second part, consider any two events X and Y where X∈ S_E and Y∈ S_E. According to the definition of S_E, the time intervals for X and Y both include the point ELT. Therefore X || Y.

After determining the set of concurrent events that will be processed in this iteration of the algorithm, Algorithm 1 computes a second global value, the latest E-time (LET) among the events in S_E. This value is also shown in Figure 1(b). The LET value is used to set the E-time of the simulation clock in each LP. The clock interval [LET,ELT] represents the intersection of the time intervals of events in S_E. This property is captured by the following lemma.

Lemma 2. Consider a non-empty set of events S with each event assigned a time interval. Let ELT be the minimum L-time among the events in S. Let S_E be the set of events {X: E(X) ≤ ELT ≤ L(X)}. Let LET be the maximum E-time among the events in S_E. Then all points in the interval [LET, ELT] must be included in the interval for each event in S_E.

Proof. We prove this lemma by contradiction. Suppose some point T exists where LET ≤ T ≤ ELT, and T is not included in the interval of some event X∈ S_E. This

means either $T<E(X)$ or $T>L(X)$. If $T<E(X)$ then $LET \leq T < E(X)$. This implies there is an event $X \in S_E$ with E-time larger than LET, contradicting the definition of LET. Similarly, if $T>L(X)$, then $L(X)<T \leq ELT$. This implies there is an event $X \in S_E$ with L-time less than ELT, contradicting the definition of ELT. Therefore, every point in [LET,ELT] is also in the time interval of each event in S_E.

The following lemma verifies that the clock values produced by algorithm 1 form a non-decreasing sequence.

Lemma 3. The sequence of E-time values stored into the simulation clock defined in algorithm 1 form a non-decreasing sequence.

Proof. This lemma can be proved by induction. The lemma is trivially true after the first iteration of algorithm 1, assuming only non-negative time values are used in the simulation. Consider the ith iteration. Let LET^i be the E-time of the clock set during the ith iteration, S^i be the set of unprocessed events at the start of the ith iteration, and S^i_E be the set of events processed during the ith iteration. S^{i+1} is equal to $S^i - S^i_E + N^i$, where N^i is the set of new events scheduled during the ith iteration. $S^i - S^i_E$ contains no events with E-time less than or equal to LET^i because all such events are processed during the ith iteration. Further, the E-time of each event in N^i must be at least LET^i, according to the AT-scheduling constraint. Thus, there will be no events in S^{i+1} with E-time less than LET^i. Thus, LET^{1+1} must be at least as large as LET^i.

We observe that at the end of each iteration of algorithm 1, the E-time of the simulation clock is equal to the maximum E-time among all events that have been processed thus far in the parallel simulation.

We are now ready to verify the correctness of algorithm 1, i.e., the algorithm does process events in AT-order, and no LP processes an event in its past.

Theorem 1. Algorithm 1 guarantees that no LP will process events out of AT-order. Further, no LP will process an event X such that X~>C, where C is the LP's clock value when the event is processed.

Proof: Let $[LET^i, ELT^i]$ denote the simulation clock defined during the ith iteration of the algorithm. We prove this theorem by contradiction. Suppose some event X is processed during the ith iteration of the algorithm, and X~>Y for some event Y processed either in this or an earlier iteration of the algorithm. Since all events processed during each iteration are concurrent (lemma 1), Y could not have been processed during the ith iteration, so Y must have been processed in an earlier iteration. Assume Y as processed in iteration h where h < i. Observe that the time interval of each event processed during any iteration of the algorithm must include the LET value for that iteration (lemma 2). Thus $E(X) \leq LET^i \leq L(X)$ and $E(Y) \leq LET^h \leq L(Y)$. Because X~>Y, L(X)<E(Y). This implies $LET^i < LET^h$. This contradicts lemma 3, however, that says that LET values are non-decreasing. Therefore, events are processed in algorithm 1 in AT-order.

The second part of this theorem, that no event X is processed where X ~> C where C is the current clock value, follows immediately from lemma 2. This lemma shows that X || C for each event that is processed.

This theorem verifies the correctness of Algorithm 1. While restricting concurrent processing to events containing exactly the same time stamp is normally considered too restrictive in time stamp order based simulations, this constraint may not be unreasonable for AT simulations. Performance depends on the number of concurrent events, which in turn depends on the number of pending events each iteration and the width of the time intervals.

4.3.2 An Asynchronous AT-Order Synchronization Algorithm

The principal virtue of algorithm 1 is its simplicity. This algorithm can be improved in several ways, however:

- The algorithm requires two global reduction computations each iteration to compute ELT and LET values. The LET computation can be eliminated by simply setting the simulation time clock to ELT rather than the [LET, ELT] interval.
- The algorithm does not attempt to exploit lookahead information that may be available from the simulation. Further, if a minimum time interval size is defined, this can also be exploited, as will be seen momentarily.
- An asynchronous version of the algorithm is preferable in order to reduce the amount of time LPs spend waiting for other LPs to reach the synchronization point.

The AT-order synchronization algorithm must ensure events are processed in AT-order, and an LP does not receive an event with time interval that precedes the LP's current time. These objectives can be met by computing the following value for each LP:

Lower Bound on Late-Time (LBLT). The $LBLT_i$ for LP_i is a lower bound on the L-time value of any future message that could later be received by LP_i.

Note that LBLT is different from ELT because ELT is the minimum L-time of any event in a snapshot of the simulation, but LBLT is a lower bound on the L-time of future messages that may be received. The LBLT value for an LP serves a purpose similar to the lower bound on the time stamp (LBTS) of messages an LP may receive in the future in a parallel time stamp order simulation. Specifically, the synchronization protocol can guarantee that LP_i does not receive messages in its past by ensuring $C_E \leq LBLT_i$, where C_E is the E-time of the LP's simulation clock. Similarly, one can guarantee the LP processes events in AT-order by requiring that $E(X) \leq LBLT_i$ for any event X processed by the LP.

Existing synchronization algorithms for time stamp ordering can often be adapted to compute LBLT values. One such algorithm, is described next. This algorithm uses concepts similar to those used in conservative synchronization and GVT algorithms such as those described in [13-16]. It assumes any LP can send a message to any other LP. A single lookahead value is associated with LP_i, denoted L_i. Also, it is assumed all messages sent by LP_i have a minimum time interval size of at least $MTSI_i$, i.e., L(X)-

$E(X) \geq MTSI_i$ for each message X sent by the LP_i. Either or both of L_i and $MTSI_i$ may be zero. In addition to exploiting lookahead and minimum time stamp interval sizes, this algorithm is asynchronous, i.e., LBLT computations proceed in background, as needed, and no global barriers are required, although a global, GVT-like computation is used. In the following, the LP_i's simulation time clock is denoted $[CE_i, CL_i]$.

$LBLT_i$ values are computed using the following information:

- $LTOut_i$: a lower bound on the L-time of future outgoing messages LP_i may generate. $LTOut_i$ is computed as $CE_i + L_i + MTSI_i$. This value is used in the LBLT computation if the LP is not blocked, waiting for simulation time to advance.
- $CLTOut_i$: a conditional lower bound on the L-time of future outgoing messages LP_i may generate, assuming the no additional messages are passed to LP_i from other LPs. Typically, this value will be computed as *min(E(X))* + L_i + $MTSI_i$ where *min(E(X))* is the minimum E-time among the unprocessed events in LP_i. The larger of $LTOut_i$ and $CLTOut_i$ is used in the LBLT computation if the LP is blocked, waiting for LBLT to advance.
- L(X) for transient messages X: the L-time of messages that have been sent, but have not been received while the LBLT computation is being performed. More precisely, $LBLT_i$ can be computed as the minimum value along a consistent cut of the distributed computation [15]. Transient messages are those that cross the cut.

$LBLT_i$ can be computed as the minimum among the above values in a consistent cut of the distributed computation. This computation is similar to that used to compute GVT in optimistic simulations. This value gives the simulation executive sufficient information to deliver events to the LP and to advance its clock.

5 Approximate Time Causal Order

One limitation of AT-ordering is it may not correctly order causal relationships in the distributed simulation. Consider the scenario depicted in Figure 3. LP_1 is modeling a tank that schedules an event to indicate it is firing upon a target. After receiving this

event, LP_2, modeling the target, schedules a new event indicating it has been hit. Both events are sent to a third, observer LP. If the "fire" and "hit" events are assigned overlapping time intervals, they may be processed in any order, but the fire event must precede the hit event because there is a causal relationship between them. While one could force a particular ordering by assigning appropriate, non-overlapping time intervals, this can become cumbersome, particularly if causal relationships are spread over LPs on different processors.

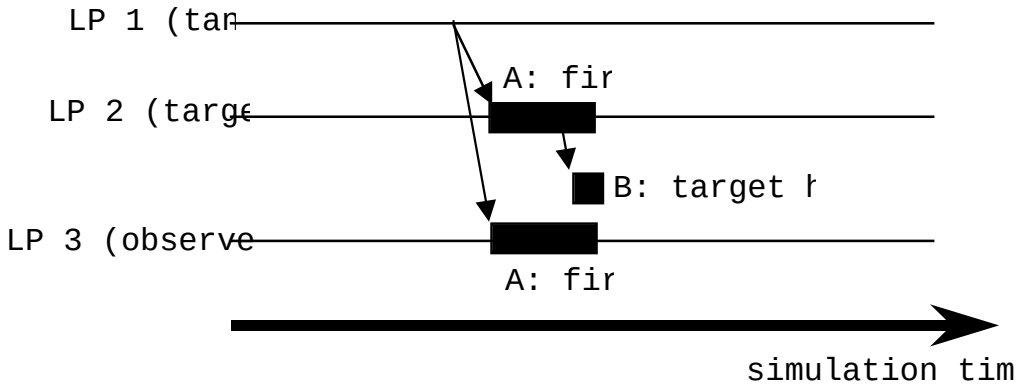

Figure 3. Ordering of causally related events.

5.1 Event Ordering

An alternate solution is to strengthen the ordering mechanism to ensure proper ordering of causal relationships. For this purpose we define AT-causal (ATC) ordering. ATC ordering is similar to AT ordering, except causal ordering is imposed on the ordering of events containing overlapping time intervals.

Causal ordering is based on Lamport's happens before ($\rightarrow$) relationship [17]. The computation in each logical process is viewed as a sequence of actions. Here, an action is a computation that processes an event, sending a message, or receiving a message. The happens before relationship is defined as follows:

- If X and Y are two actions within the same LP, and X takes place before Y, then $X \rightarrow Y$ (read X happens before Y).
- If X is sending a message M and Y is receiving M, then $X \rightarrow Y$.

- If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$ (transitivity).

Each event X is assigned a time interval [E(X), L(X)], as was the case in AT-ordering. AT-causal ordering (represented $\sim>_C$) is defined as follows:

- If for two events X and Y L(X) < E(Y), then $X \sim>_C Y$.
- If X and Y have at least one point in common, and $X \rightarrow Y$, then $X \sim>_C Y$.
- If neither of the above constraints apply, then X and Y are concurrent events (X || Y).

For example, in Figure 3 $A \sim>_C B$ because $A \rightarrow B$ and the two events have overlapping time intervals.

5.2 ATC Simulations and Synchronization Algorithms

ATC simulations follow the same principles as AT simulations, except ATC is used as the ordering mechanism rather than AT order. Specifically, each LP in an ATC simulation must process events in ATC order. Further, an LP should never receive an event with a time interval preceding that of the LP's current time interval.

It is clear that the principal new requirement that must be satisfied to realize an ATC simulation compared to an AT simulation is ensuring that concurrent AT-order events are processed in causal order. An extensive literature is available discussing methods for implementing causal ordering that can be brought to bear on this problem. A simple technique for implementing causal ordering is to use global synchronization points. Specifically, causal ordering can be guaranteed by forcing all concurrent events to be processed before any of the new events produced as a result of processing these events are delivered to LPs. Algorithm 1 described earlier for implementing AT-order simulations can thus be used to realize ATC simulations provided one ensures successive concurrent messages sent by one LP to another are delivered in the same order in which they were sent. The barrier synchronization at the end of each iteration ensures causal ordering of concurrent events. For example, in Figure 3 both instances of the fire event will be processed in one iteration, and the hit event will be processed in a later iteration.

6 Experimental Evaluation

An implementation of the AT-order algorithm has been developed to evaluate the effectiveness of AT-order simulations. The principal questions investigated in this study are to evaluate the performance benefits that can be achieved by introducing time intervals, and to evaluate the impact of time intervals on the accuracy of the simulation results that are produced.

6.1 The Approximate Time RTI

A principal application of AT and ATC ordering mechanisms is in developing efficient federations of simulations, e.g., distributed simulation systems realizing the High Level Architecture. In particular, a goal of these initial performance studies was to evaluate the effectiveness of this approach compared to time stamp order synchronization techniques in federating simulations with zero lookahead, without resorting to optimistic processing techniques. The simulation consists of a collection of federates, where each federate is a stand alone sequential simulator augmented to interface it to the distributed simulation system.

The central ramification of this approach is it means internally, each federate implements a traditional event-oriented execution mechanism using *precise* time stamps. Approximate time ordering is only used for messages transmitted between federates. The federate assigns a time interval to each event sent to other federates, but local events that remain within the federate are only assigned a precise time stamp. Further, incoming messages are assigned a precise time stamp that lies within the message's time interval before they are passed to the federate. Each federate then process all events, both locally and remotely generated, in time stamp order. The clock for each federate is defined as a precise time value.

The Approximate Time RunTime Infrastructure (AT-RTI) was developed as the distributed simulation executive for this work. This RTI implements a portion of the interface specification for the High Level Architecture [18], modified to implement the

approximate time ordering services. In particular, precise time stamps defined in the I/F Spec are replaced with time intervals.

In addition to assigning a time interval to each event that is sent between federates, the sender also assigns a *target time stamp*. This value indicates the time stamp the federate would have assigned to the event had precise time stamps been used. This information is used when a time stamp is later assigned to the event by the receiving federate. In particular, before each message is delivered to the federate, the AT-RTI makes a call back to the federate specifying a range of *allowed time values* that may be assigned to the event (a subset of the time interval originally assigned to the event) and the target time stamp value. The federate returns the precise time value that it wishes to assign to the event. The AT-RTI then delivers events to the federate in time stamp order. In the experiments described here the federate uses the target time stamp if it is within the allowed time interval, and the closest available time stamp if it is not.

The asynchronous version of the AT-order synchronization algorithm described earlier was used in this initial implementation. An asynchronous distributed snapshot algorithm is used to compute LBLT values. This algorithm and its implementation are described elsewhere [19]. Key features of this algorithm are summarized below.

- The algorithm is asynchronous, and executes "in background" concurrent with other simulation computations, and does not require barriers.
- The algorithm computes a global minimum along a consistent cut of the distributed computation.
- A butterfly reduction network is used to compute global minimum values, allowing a global minimum, and distribution of the minimum value to each of N processors to be completed in log N steps, in the absence of transient messages.
- A coloring scheme is used to detect transient messages.
- Counters of the number of sent and received messages are used to determine when all transient messages have been received, and have been accounted for in the global minimum computation.

The AT-RTI on each processor initiates a new LBLT computation whenever the federate on that processor requests its simulation time be advanced (via a service similar to the HLA Next Event Request service), but the request cannot be granted and no events are available for delivery to the federate. If more than one processor simultaneously initiates a new LBLT computation, the AT-RTI software automatically combines these initiations into one LBLT computation. In the experiments described here, a processor will not initiate a new LBLT computation if one is already in progress, although the LBLT software supports multiple, concurrent LBLT computations to be in progress at one time. If the LBLT computation does not return a value that is sufficiently large to complete the federate's request to advance simulation time, a new LBLT computation is initiated. The federate and RTI are free to perform other computations, e.g., new messages may arrive, while an LBLT computation is in progress.

6.2 Cluster Computing Environment

A cluster of workstations was used for these experiments. In particular, eight Sun Ultrasparc-1 workstations interconnected by a low latency Myrinet switch were used. The workstations run Solaris version 5.5. LANai Version 4.1 Myrinet cards with 640 MBit/second communication links are used. Newer 1.28 GBit switches are currently available from Myricomm, but were not used in these experiments. Myrinet switches provide point-to-point switched interconnection between workstations, and feature cut-through message routing to reduce message latency [20].

The AT-RTI software was developed on top of version 1.3 of the RTI-Kit software package [19]. RTI-Kit is a collection of libraries implementing key mechanisms that are required to realize distributed simulation RTIs, especially RTIs based on the High Level Architecture. The RTI-Kit libraries are designed so they can be used separately, or together to enhance existing RTIs, or to develop new ones. The AT-RTI uses RTI-Kit's MCAST library that provides group communications facilities, and the TM-Kit library that implements time management functions. The version of RTI-Kit used in this study is built over version 2.0 of the Fast Message software package developed at the University

of Illinois [21]. Other implementations of RTI-Kit are built over TCP/IP, and over shared memory in SMP environments, but were not used in this study.

Federate-to-federate communication latency using the RTI-Kit software was measured at 20 to 30 microseconds for messages of 128 bytes or less on the Sun/Myrinet configuration. A global reduction operation requires about 20 microseconds for two processors, and 70 microseconds for eight. Lower latency numbers can be expected on newer hardware configurations, e.g., newer workstations and the 1.28 GBit Myrinet switch.

6.3 Applications

Here, we are particularly interested in simulations with zero lookahead, because it is well known that conservative simulation techniques fail for this important class of applications. Both of the benchmark simulations used in this study are well-known benchmarks widely used in the parallel discrete event simulation community.

The first benchmark is the PHOLD synthetic workload program [22]. A single logical process is mapped to each federate, and LPs do not send messages to themselves. Thus, there are no local events, i.e., each event processed by a federate was generated on a different processor. The target time stamp is selected from an exponential distribution with mean 1.0, and the destination federate is selected from a uniform distribution. The minimum time stamp increment is zero, resulting in a zero lookahead simulation.

The second application is a queueing network, configured in a toroid topology. This simulation does *not* attempt to exploit lookahead. A textbook approach to realizing this simulation was used that includes both job arrival and job departure events. Each departure event schedules an arrival event at another queue with time stamp equal to the current simulation time of the LP scheduling the event, i.e., messages sent between LPs all contain zero lookahead. Service times are selected from an exponential distribution, and jobs are serviced in first-come-first-serve order. Each federate models a rectangular portion of the queueing network.

It is well known that certain classes of queueing network simulations can be optimized to exploit properties of the queues to improve their lookahead. This approach was intentionally *not* used here, because such optimizations lead to fragile code that is highly dependent on specific aspects of the model. Such techniques are generally difficult to apply in large, complex models. Rather, the experiments performed here were intended to evaluate the effectiveness of approximate time ordering in federating simulations in distributed computing environments without rewriting the models to exploit lookahead.

6.4 Execution Speed

A sequence of experiments was performed using approximate time ordering where the interval size was varied. Each experiment was repeated five times, using different random number generator seeds in each execution. All messages sent between federates use the same interval size, which was also used as the MTSI value. As mentioned earlier, all simulations have zero lookahead. The data point with zero interval size corresponds to the case where precise time stamps and time stamp order event processing are used.

The PHOLD simulation uses one logical process per federate, and one federate on each of the eight processors. Simulations with a job population of 256 and 4096 were performed. The queueing network simulations modeled 256 (16 by 16) and 4096 (64 by 64) queues, and in each case assumed fixed populations of 256 and 4096 jobs, respectively, or one job per queue.

The performance of the simulation is plotted relative to the distributed simulation using precise time stamps in Figure 4. As can be seen, increases in the interval size lead to significant improvements in performance, more than a twenty fold reduction in execution time in some cases. Even modest sized time intervals lead to significant performance improvements. For example, an interval size of 0.1 for the PHOLD application implies the time stamp increment is set to $X \pm 0.05$, or $\pm 5\%$ in most cases since the average time stamp increment is 1.0.

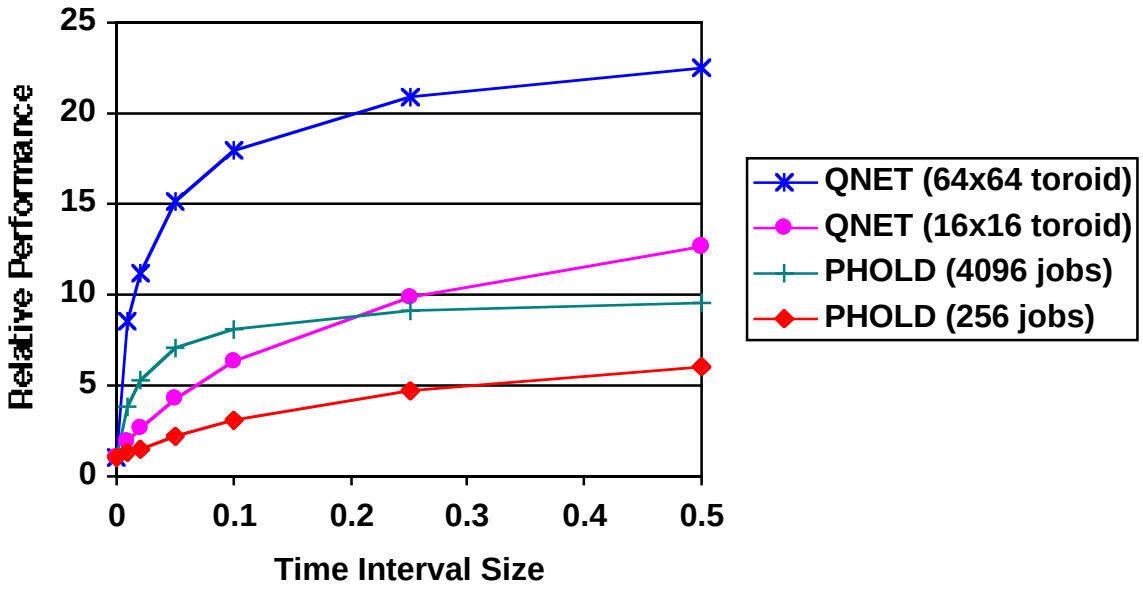

Figure 4. Execution Speed using Approximate Time Ordering.

The performance improvement can be attributed to reduced synchronization overheads because of the use of time intervals. In particular, because these simulations contain zero lookahead, a synchronization step is required after virtually every event in a time stamp ordered simulation. Time stamp intervals allow events that occur at approximately the same time to be processed concurrently, greatly reducing the number of synchronization operations that must be performed. This is the principal reason interval sizes lead to improved performance.

The performance data in Figure 4 also indicates greater performance improvement occurs for larger simulation models. This is because for large models, there are more unprocessed events, so more events will have overlapping intervals, leading to increased concurrency. In general, AT order can be expected to yield greater concurrency as the number of pending events in the model at one time increases.

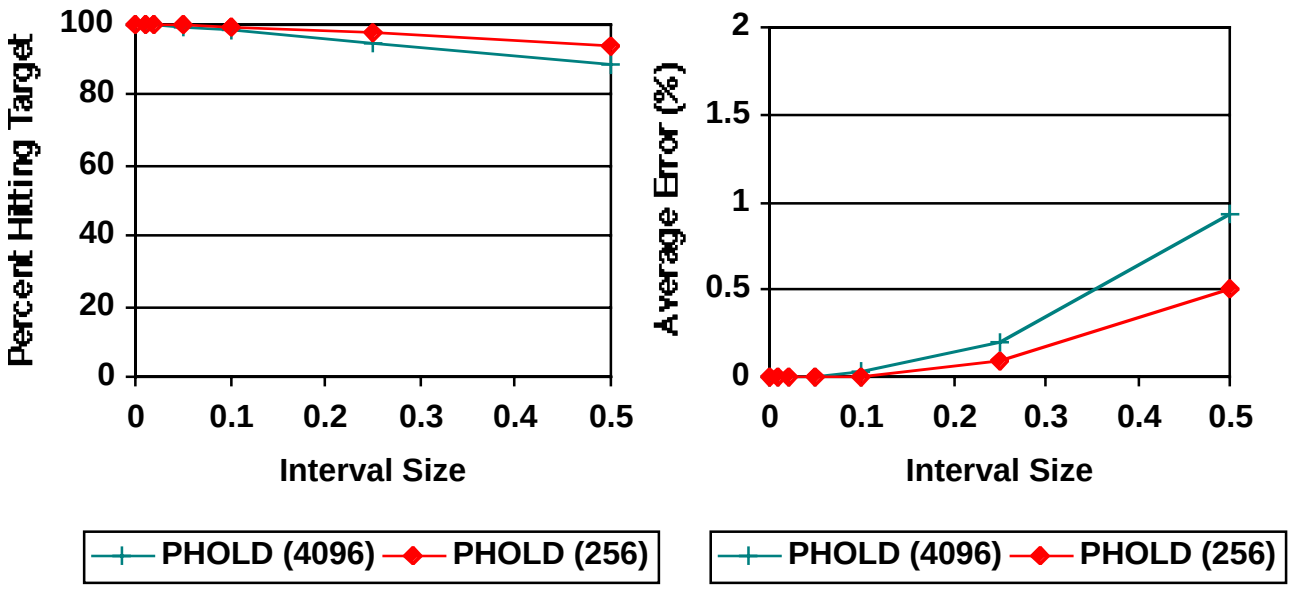

Figure 5. Accuracy of the PHOLD simulation. (a) fraction of assigned time stamps hitting taget. (b) Average error in time stamp assignment.

6.5 Accuracy

The PHOLD and queueing network simulation computations were examined in order to assess the impact of using time intervals on the results computed by the simulation. In PHOLD, the time interval is selected so that the target time stamp is in the middle of the interval, except in cases where this would cause the beginning of the interval to precede the current time of the federate. In this case, the interval was shifted to begin at the federate's current logical time. Recall the federate's current time was defined as a precise time value in these experiments.

The simulation was instrumented to count the number of times the actual time stamp assigned to an event exactly matched the target time stamp. The fraction of times the target time stamp was assigned to the event is shown in Figure 5(a). For interval sizes up to 0.1 (± 0.05) over 98% of the assigned time stamps hit the target, and the average error

was less than 0.0003 (see Figure 5(b). Even with intervals as large as 0.5 (± 0.25) almost 90% or more of the assignments hit the target, and the average error was less than 1%.

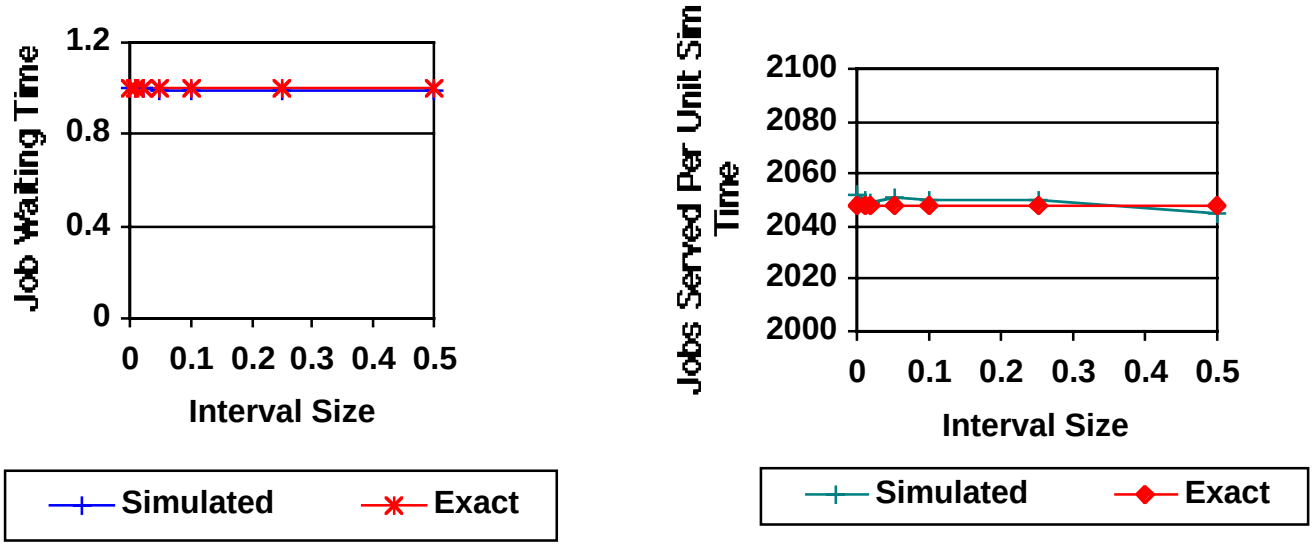

Figure 6. Accuracy of the queueing network simulation. (a) mean job waiting time. (b) jobs processed per unit simulation time.

The queueing network simulations computed two statistics: the average waiting time encountered by a job when it arrived at a station, and the average number of jobs served in the network per unit of simulation time. Note these are statistics computed by the queueing network simulation, *not* parallel performance statistics. Here, simulation results could be compared against the exact solution because these queueing networks have simple closed form solutions. Data for the larger (64 by 64) queueing network is shown in Figure 6. The data for the smaller network showed similar behaviors. Figure 6(a) shows the average job waiting time for different interval sizes as well as the exact solution. In all simulation runs the simulator predicted average waiting times within 0.9% of the exact solution. Figure 6(b) shows the total number of jobs serviced per unit simulation time. In all runs the simulated results were within 0.4% of the exact solution. These results indicate that the use of relaxed ordering has virtually no impact on the

numerical results computed by the simulation for the models and statistics that were tested.

7 Conclusions and Future Work

The zero lookahead problem has been a long standing problem in parallel and distributed simulation systems. This problem is particularly acute in federated simulation systems because the time and expense required to introduce rollback into the simulation program often precludes use of optimistic synchronization. In addition, the historical dichotomy that has developed between parallel and distributed discrete event simulation techniques and work in distributed virtual environments (e.g., DIS) has made reuse of simulation models across these domains difficult.

Approximate time and approximate time causal ordering mechanisms provide an alternate approach to distributed simulation. Approximate time order includes time stamp order and DIS style unordered communications as special cases. Initial experiments indicate that this approach shows promise in achieving good parallel performance of zero lookahead simulations without resorting to optimistic processing techniques. The performance gains were realized with negligible impact on the numerical results produced by the simulator for the test applications that were examined.

The results described here only scratch the surface in the use of AT and ATC ordering for parallel and distributed simulations. Numerous issues remain to be investigated. A few are listed below:

- Model validation. What size time intervals can be used without invalidating the simulation model? While traditional model validation methods can be applied to at least partially address this question, more work is clearly needed in this area.
- Application studies. More extensive investigation of the use of these ordering mechanisms in real world applications is required, including evaluation of the impact of relaxing ordering constraints on the results produced by the simulation.
- Repeatability. Additional mechanisms are required to ensure repeated executions of the simulation with the same input and initial state produce the same results.

- Time stamp assignment. In simulations requiring precise time stamps (e.g., federating conventional simulations), the question of what time stamp to assign each event requires further study. Our current implementation uses simple rules that do not attempt to minimize error. Another problem is the case where there are multiple recipients of messages for the same event. It may be necessary to ensure the same time stamp is assigned to each message. Additional mechanisms are required to ensure that this is the case.
- Synchronization algorithms. More advanced algorithms can improve the efficiency of the AT and ATC ordering mechanisms. In particular, algorithms that exploit communication topology have not yet been examined.
- Use in wide area networks. Approximate time synchronization provides a means for relaxing synchronization constraints, thereby reducing overheads necessary to ensure the correct operation of a simulation. Such overheads are particularly onerous in wide area networks where communication latencies dominate. A potential application of relaxed synchronization constraints such as that proposed here is in realizing distributed simulation systems over geographically distributed locations.

While many open questions remain, the initial results using AT and ATC order reported here are encouraging. Thus, a principal conclusion of this preliminary work is time intervals offer excellent potential for synchronizing distributed simulations, especially federated simulation systems.

8 References

1. Miller, D.C. and J.A. Thorpe, *SIMNET: The Advent of Simulator Networking.* Proceedings of the IEEE, 1995. **83**(8): p. 1114-1123.

2. IEEE Std 1278.1-1995, *IEEE Standard for Distributed Interactive Simulation -- Application Protocols*. 1995, New York, NY: Institute of Electrical and Electronics Engineers, Inc.

3. Wilson, A.L. and R.M. Weatherly, *The Aggregate Level Simulation Protocol: An Evolving System*, in *Proceedings of the 1994 Winter Simulation Conference*. 1994. p. 781-787.

4. Nicol, D.M. and P. Heidelberger, *Parallel Execution for Sequential Simulators.* ACM Transactions on Modeling and Computer Simulation, 1996. **6**(3): p. 210-242.

5. Fujimoto, R.M., *Parallel and Distributed Simulation Systems.* 1999: Wiley Interscience.

6. Fujimoto, R.M., *Performance Measurements of Distributed Simulation Strategies.* Transactions of the Society for Computer Simulation, 1989. **6**(2): p. 89-132.

7. Sokol, L.M. and B.K. Stucky, *MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm*, in *Proceedings of the SCS Multiconference on Distributed Simulation*. 1990. p. 169-173.

8. Porras, J., J. Ikonen, and J. Harju, *Applying a Modified Chandy-Misra Algorithm to the Distributed Simulation of a Cellular Network*, in *Proceedings of the 12th Workshop on Parallel and Distributed Simulation.* 1998, IEEE Computer Society Press. p. 188-195.

9. Allen, J., *Maintaining Knowledge about Temporal Intervals.* Communications of the ACM, 1983. **26**(11): p. 832-843.

10. Dubois, D. and H. Prade, *Processing Fuzzy Temporal Knowledge.* IEEE Transactions on Systems, Man, and Cybernetics, 1989. **19**(4): p. 729-744.

11. Birman, K., A. Schiper, and P. Stephenson, *Lightweight Causal and Atomic Group Multicast.* ACM Transaction on Computer Systems, 1991. **9**(3): p. 272-314.

12. Raynal, M. and M. Singhal, *Logical Time: Capturing Causality in Distributed Systems.* IEEE Computer, 1996. **29**(2): p. 49-56.

13. Groselj, B. and C. Tropper, *The Time of Next Event Algorithm*, in *Proceedings of the SCS Multiconference on Distributed Simulation*. 1988, Society for Computer Simulation. p. 25-29.

14. Chandy, K.M. and R. Sherman, *The Conditional Event Approach to Distributed Simulation*, in *Proceedings of the SCS Multiconference on Distributed Simulation*, B. Unger and R.M. Fujimoto, Editors. 1989, Society for Computer Simulation. p. 93-99.

15. Mattern, F., *Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation.* Journal of Parallel and Distributed Computing, 1993. **18**(4): p. 423-434.

16. Nicol, D.M., *Noncommittal Barrier Synchronization.* Parallel Computing, 1995. **21**: p. 529-549.

17. Lamport, L., *Time, Clocks, and the Ordering of Events in a Distributed System.* Communications of the ACM, 1978. **21**(7): p. 558-565.

18. Defense Modeling and Simulation Office, *High Level Architecture Interface Specification, Version 1.3,* . 1998: Washington D.C.

19. Fujimoto, R.M. and P. Hoare, *HLA RTI Performance in High Speed LAN Environments,* in *Proceedings of the Fall Simulation Interoperability Workshop*. 1998: Orlando, FL.

20. Boden, N., *et al.*, *Myrinet: A Gigabit Per Second Local Area Network.* IEEE Micro, 1995. **15**(1): p. 29-36.

21. Pakin, S., *et al.*, *Fast Message (FM) 2.0 Users Documentation*, . 1997, Department of Computer Science, University of Illinois: Urbana, IL.

22. Fujimoto, R.M., *Performance of Time Warp Under Synthetic Workloads,* in *Proceedings of the SCS Multiconference on Distributed Simulation*. 1990. p. 23-28.