Cloning Parallel Simulations

MARIA HYBINETTE
The University of Georgia
and
RICHARD M. FUJIMOTO
Georgia Institute of Technology

We present a cloning mechanism that enables the evaluation of multiple simulated futures. Performance of the mechanism is analyzed and evaluated experimentally on a shared memory multiprocessor. A running parallel discrete event simulation is dynamically cloned at *decision points* to explore different execution paths concurrently. In this way, what-if and alternative scenario analysis can be performed in applications such as gaming or tactical and strategic battle management. A construct called *virtual logical processes* avoids repeating common computations among clones and improves efficiency. The advantages of cloning are preserved regardless of the number of clones (or execution paths). Our performance results with a commercial air traffic control simulation demonstrate that cloning can significantly reduce the time required to compute multiple alternate futures.

Categories and Subject Descriptors: I.6.1 [**Simulation and Modeling**]: Simulation Theory; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Discrete event*; *parallel*; *distributed*; D.4.1 [**Operating Systems**]: Process Management—*Concurrency*; *scheduling*; *synchronization*; D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed programming*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Cloning, multiprocessors, parallel algorithms, parallel simulation, pruning

1. OVERVIEW

Simulation is becoming an important tool for decision makers in time-constrained environments. High-fidelity simulations demand parallel implementations to support the decision process at real-time rates. Typical decision making applications in which parallel simulations can or do play a role include air traffic control, gaming strategy and battle management among others. The goal of this research is to provide a simulation-based decision aid for managers

This work was supported in part by U.S. Army Contract DASG60-95-C-0103 funded by the Ballistic Missile Defense Organization and managed through the Space and Strategic Defense Command, a MITRE Corporation research grant, and a grant from Sun Microsystems.

Authors' addresses: Maria Hybinette, Computer Science Department, The University of Georgia, Athens, GA 30602-7404; email: maria@cs.uga.edu; Richard M. Fujimoto, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280; email: fujimoto@cc.gatech.edu.

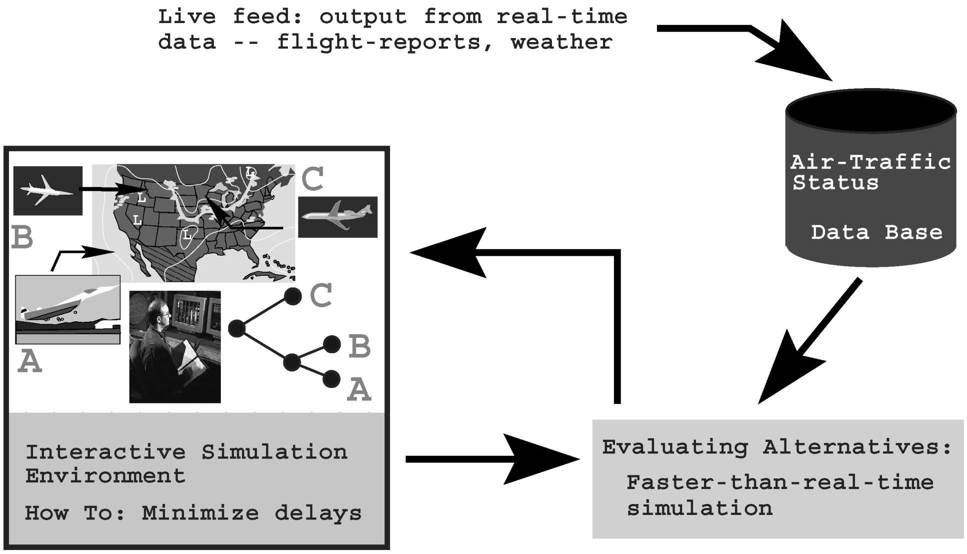
Live feed: output from real-time data -- flight-reports, weather
Air-Traffic Status Data Base
B
C
C
B
A
A
Interactive Simulation Environment
How To: Minimize delays
Evaluating Alternatives:
Faster-than-real-time simulation

Fig. 1. Envisioned system: A faster-than-real-time simulation is integrated in a real-time interactive environment, enabling what-if evaluation of alternative causes of action.

faced with complex planning tasks (e.g., real-time air traffic management). To realize effective decision making we propose a technology called *simulation cloning* that enables more efficient exploration of different possible future outcomes based on policy decisions made at well defined *decision points*.

The envisioned decision aid includes three components: (1) a situation data base built from live data feeds that characterizes the current state of the system under investigation, (2) a faster-than-real-time forecasting simulator and (3) an interactive environment that interfaces with both the database and the forecasting tool for what-if evaluation of alternative courses of action. The overall system is depicted in Figure 1 in the context of an air traffic control application.

A powerful simulation-based decision tool should provide capabilities such as monitoring and steering. Monitoring provides the ability to query or sample simulation variables. These variables may be sampled continuously, displayed when a pre-defined condition is satisfied or presented on demand.

Simulation steering controls the execution of an application, such as pause, execution path modification and rollback. Pausing stops the simulation and enables the analyst to fully investigate the state of the simulation to determine whether to to create new execution paths or to modify variables of the current execution path. Execution path modifications change the state of the simulation and are added to the simulation at *decision points*. Rollback is the retraction to an earlier simulated time and provides for the re-evaluation of old decision points and variables.

Such a tool could be used for example, to determine which of several air traffic control decisions offers minimal delay. If thunderstorms are approaching a controller may want to evaluate alternative air traffic restrictions to minimize travelers' delays while maintaining safety in the airspace. This can be done by dynamically cloning (replicating) several faster-than-real-time simulations with different courses of action. The clones utilize both the current situation and forecasting parameters such as the start time of the restriction. The clones are inserted as decision points and each of the new "clones" proceeds in parallel, sharing common computations to improve performance (sharing a computation means that the computation is performed only once among the clones). For example, the clones will share all computations before the start time of the restriction. The effect of each policy within each clone is monitored to determine which policy offers the most effective solution. This research focuses on the details of the parallel evaluation of multiple alternatives.

Parallel discrete event simulation techniques are used to speed up the execution of the simulator. The parallel simulator consists of distinct components called logical processes or LPs. Each LP models some part of the system under investigation and maintains its own state. For example in an air traffic application each airport may be modeled by an LP. The logical processes are often mapped to different processors and interact by exchanging time-stamped event messages. The simulation progresses as LPs exchange messages that model changes of the system state at discrete points in time. In the air traffic application for example, the airports may interact by sending messages modeling aircraft arrival and departure events.

In order to maintain consistency, events are processed in time-stamp order according to a consistency protocol. The two prevailing consistency protocols are termed *conservative* and *optimistic*. The conservative protocol enforces consistency by each LP processing its events in time stamp order. The optimistic protocol, in contrast, allows out of order event processing, but provides a mechanism such as rollback to recover. Our cloning mechanism is applicable to both conservative and optimistic simulation protocols.

The rest of this paper is organized as follows: Background and related work is discussed in Section 2. Section 3 introduces dynamic cloning. In Section 4 we present the virtual logical-process paradigm. This is followed by a discussion of the cloning mechanism in Section 5. We then describe our implementation of the cloning mechanism in Section 6. Next, performance measurements are presented and analyzed. We conclude with a summary and a discussion of future directions.

2. RELATED WORK

Related work in interactive parallel discrete event simulation is described in Steinman [1991] and in Franks et al. [1997]. In Steinman, a message type called an *external* event enables interaction with the simulator. These events can steer, sample and query the simulation in progress. The approach of Franks et al. allows for the testing of what-if scenarios provided they are interjected before a deadline. Alternatives are examined sequentially by using a rollback mechanism to erase exploration of one path in order to begin exploration of another. In contrast to these methods, our algorithm dynamically creates and evaluates multiple alternatives to allow concurrent exploration of multiple paths. New versions of an executing simulation can be dynamically cloned to evaluate alternative paths. Because the cloned simulation proceeds in parallel with the original, an increased number of alternatives can be evaluated.

Cloning was suggested by von Neumann [1956] more than 40 years ago to provide fault-tolerance. Fault-tolerance is ensured by providing identical processes, identical transactions, duplicate data, or redundant services [Schneider 1990]. Cloning can also improve throughput by placing process replicas in the proximity of where a service is needed [Goldberg 1992; Schneider 1990]. Cloning is suggested as a solution for concurrency control in real-time databases [Bestavros 1994] and to improve accuracy of simulation results—to run multiple independent replications then average their results at the end of the runs. For example, in Heidelberger [1991] and Glynn and Heidelberger [1991], research focuses on how to achieve statistical efficiency of replications spread on different machines. In Vakili [1992], replications are synchronized via a shared clock. In this way the same event occurs at the same time in all replicas.

Cloning is also used in sequential simulations to propagate faults in digital circuits [Lentz et al. 1999], to model flexible manufacturing systems [Davis 1998] or to fork transactions in simulation languages [Henriksen 1997]. Sequential simulation languages typically clone dummies (“shadows”) to do time-based waiting [Schriber and Brunner 1997].

The incremental update schemes of process migration algorithms such as Zayas [1987] are similar in philosophy to our virtual logical process scheme (covered in a later section). The common goal is to reduce the cost of copying the virtual address space between clones. The process migration algorithms differ from ours in that they only support one active clone, while we allow for multiple clones. Our main motivation is to develop a parallel model that supports an efficient, simple, and effective way to explore and compare alternate scenarios. This is decidedly different from the studies reported above.

3. SIMULATION CLONING

Simulation cloning enables the creation of several new copies of a running simulation, each modeling a different course of action. The clones are inserted into the on-going simulation as decision points and each of the new clones proceeds in parallel. Here, a decision point defines the point in simulation time where the simulation model is cloned so the original and cloned simulations may execute along different execution paths.

Simulation cloning improves simulation performance in two ways: First, simulations that are interactively cloned at a decision point are able to share the same execution path before the decision point, so the computations before the decision point need only be executed once. Second, after cloning, multiple versions of the simulation may be able to share a single execution of certain computations that are common between them (a mechanism for achieving this will be described in the next section). Additionally, clones can be “pruned,”—deleted—when it is apparent that the course of action they are evaluating will not be competitive with respect to others.

These capabilities provide a significant improvement over the traditional paradigm where several alternative trials of a simulation are completely replicated with each replication evaluating a single course of action. One disadvantage of this approach is that it may be difficult to predict ahead of time which alternate courses of action or simulation parameter alternatives need to be considered. In our approach however, one can interactively clone new simulations from a running simulation as needed.

As an example, consider an air traffic control application where the goal is to minimize flight delays and aircraft "circling" while meeting the air traffic demand and safety requirements. Towards this end, air traffic controllers can adjust the spacing between aircraft—the miles in trail (MIT) to control the flow of traffic in the air space. Further, controllers can impose "ground delay" programs to keep aircraft from taking off rather than have them circle near their destination, which is both costly and is less safe. Simulation can be used to evaluate the effects of imposing different MIT restrictions and/or ground delay programs.

Suppose, for example, that congestion is expected to develop at Dulles (IAD) because of a planned runway closure that afternoon. A controller might consider reducing the rate of aircraft entering the Dulles airspace. The controller might wish to consider MIT values: 10, 20 or 30 miles in the traffic corridors near Dulles to assess their impact on flight delays of inbound aircraft, and the affect of these changes on the entire national air space. Using replicated trials, the controller would have to launch four separate simulations, each using a different restriction imposed that afternoon prior to the runway closure. Then, after analyzing the result of each simulation the controller would impose the restriction which promises the least flight delays within the overall system. This scenario is depicted in the top image of Figure 2 where each horizontal line represents the execution of one of the replications.

The middle image in Figure 2 illustrates how cloning can be applied to this scenario. A decision point is set at the time when the restrictions would be imposed. Rather than repeating the execution of the simulation prior to the decision point four times, that portion of the computation is only performed once. When the simulation time reaches the decision point, three new clones are created, and each completes the execution using different MIT restrictions. It is clear that this will be much more efficient than the replicated approach.

The bottom image of Figure 2 shows another scenario, where a controller is attempting to determine the proper time at which to enforce a restriction. In this case the *decision points* are set at different simulation times corresponding to candidate times for imposing the restriction. These two examples show that cloning may increase the number of alternate simulations that can be run by reducing the amount of computational resources required.

Another benefit of dynamic cloning is that clones can be created interactively. For example as soon as the analyst sees something interesting in the ongoing simulation he may interactively create a new clone, or pause the simulation, roll back and then create a new clone. In contrast to replicated trials, alternative simulation(s) do not need to re-run from the beginning and can be spawned as soon as it is realized a new alternative is viable.

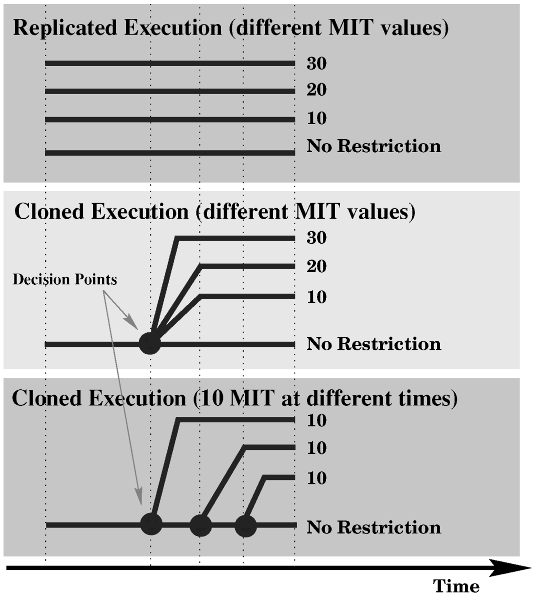
Replicated Execution (different MIT values)
30
20
10
No Restriction
Cloned Execution (different MIT values)
30
20
10
No Restriction
Decision Points
Cloned Execution (10 MIT at different times)
10
10
10
No Restriction
Time

Fig. 2. Simulations that are cloned at a decision point share the same execution path before cloning. The top image shows the current approach where four independent simulations are run separately to evaluate alternatives.

Computational resources can be even more efficiently utilized if unproductive simulation clones are *dynamically pruned*. A controller may realize that some Miles-In-Trail (MIT) restrictions will not lead to desirable outcomes long before the simulation completes. In this case the cloned simulation can be simply terminated prior to completion, releasing the resources it was using for use by other clones.

4. VIRTUAL LOGICAL PROCESSES

As previously mentioned, simulation cloning avoids the cost of simulating multiple futures by *sharing* common computations (i.e. computation that is common to more than one clone need be performed only once). In the previous section we illustrated how clones share computations *before* the branching point by defining decision points. In this section we introduce the construct *virtual logical processes*, that enables sharing of computations *after* the decision point.

Cloning can be implemented by simply replicating the entire simulation at each decision point. For large simulations, consisting of thousands or millions of LPs, this approach may be wasteful, especially when the differences among the clones are confined to only a few LPs. In this case, as the simulation progresses, many computations and messages in the different clones will be the same. It would clearly be desirable to perform these duplicate computations only once. For example, consider a decision point introducing an MIT restriction near Dulles. The new restriction will initially only affect the air space in the Washington, D.C. area. Simulation computations corresponding to more distant airports, for example Los Angeles, will be the same in the different clones until

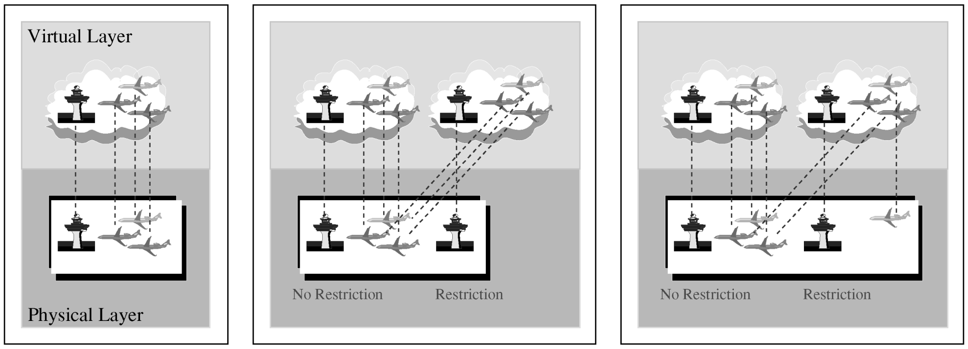

Fig. 3. An air traffic control simulation is divided into two layers: A physical layer and a virtual layer. The incremental update scheme allows sharing of computational resources until interaction.

the effects of the MIT restrictions at Dulles propagate across the national airspace. Thus the computations corresponding to LPs representing airports in the Los Angeles area need only be performed once, and their results can be shared among the different clones.

Our approach to avoiding duplicate computations is to replicate the state space of the simulation incrementally (i.e. LPs can be replicated incrementally) rather than all at once. To explain this approach, we use concepts analogous to those used in virtual memory systems. Each simulation clone is referred to as a *version* of the simulation, and consists of a set of *virtual logical processes* that communicate by exchanging *virtual messages*. As implied by their name, virtual LPs and messages do not have an actual, physical realization. Instead, each virtual LP maps to (exactly) one *physical logical process* that does have a physical realization in the parallel simulation system. Similarly, each virtual message maps to a single physical message. Common computations among two or more clones can be shared by mapping virtual LPs among the different clones to a single physical LP that performs the actual (shared) computation. Similarly, virtual messages among different clones can be mapped to a single physical message to avoid duplication.

When a new version (clone) of the simulation is created, the entire set of virtual logical processes making up the version is replicated. Replicating a virtual LP requires very little computation, however, because one need only update a few internal data structures. One need only replicate the *physical* LPs that are different in the two clones. Replicating a physical LP requires replication of the state variables included in the LP. Virtual LPs that are identical in the two clones map to the same physical LP, while those that are different in the clones map to different physical LPs.

To illustrate these concepts, again consider an air traffic control simulation. Suppose there are three aircraft approaching Dulles (IAD) airport. This scenario is depicted on the left of Figure 3. The tower represents the control center at the Dulles airport. The virtual LPs are depicted in the section of the diagram labeled the *virtual layer*, and the physical LPs are in the area labeled the *physical layer*. Initially (the leftmost diagram in Figure 3) there is a single version of the simulation, and each virtual logical process maps to a single physical logical process.

Suppose a clone of the simulation is created halfway through the execution that introduces a new MIT restriction at Dulles. The original version of the simulation does not include this change. This is shown in the middle image of Figure 3. There are now two versions of the simulation in the virtual layer. Notice that there are two physical LPs modeling the tower. One simulates actions at the tower with the new restrictions and the other represents the original tower with no new restrictions. The virtual logical processes modeling the tower at Dulles are mapped to different physical logical processes. The simulation models of the three aircraft are identical in both versions (for now) because they are not yet affected by the restriction, so they share the same physical simulation. The two virtual logical processes for each aircraft are mapped to the *same* physical logical process. In this way, common computations among the two versions need only be performed once.

Common message send and receive computations are also shared among clones. In general, when a physical LP (e.g. the aircraft LPs in Figure 3) sends a message, this corresponds to a set of message sends in the virtual layer. In the example (Figure 3), a message sent from the physical logical process corresponding to the top aircraft, to the physical logical process corresponding to the bottom aircraft, results in *two* virtual messages sent from their corresponding virtual logical processes; one message sent for each version of the simulation. This illustrates how the virtual/physical abstraction allows common communication among different clones to be shared.

To continue with our example, suppose the topmost aircraft calls the tower. In the simulation, this corresponds to a message send from an aircraft LP to the tower LP. The same communication occurs in the original execution as well as the clone. Both versions of the sender of the message (the airplane) are represented by the same physical process but there are two distinct physical processes representing the receiver (the tower). Thus, two copies of the message must be sent, one to each physical LP. To handle this situation, the cloning mechanism must duplicate, or clone the message.

Our cloning mechanism increases efficiency by delaying the cloning of physical LPs until it is absolutely necessary. As an example, eventually, as the aircraft draw closer to Dulles, the restriction impacts the aircraft. At that time, the physical layer clones the physical LP corresponding to the delayed aircraft. This is illustrated in the rightmost image of Figure 3. There are now two versions of the aircraft in the physical layer and two versions of the aircraft in the virtual layer. Each copy of the aircraft in the virtual layer maps to one version in the physical layer. Note that remaining aircraft in the virtual layer continue to share the same physical LPs in the physical layer.

5. THE CLONING MECHANISM

As mentioned earlier, LPs can only affect other LPs in a parallel simulation by exchanging messages. Thus, the reception of a message is the point at which a logical process must consider message forwarding and process cloning.

Message sends and receives are implemented in the physical layer. The cloning mechanism first determines which physical LPs should receive the message. Next it considers whether the generation, or cloning, of a new physical process is required. Whether a message is forwarded or a process is created, depends on the set of virtual processes mapped to the sending physical process and on the set of virtual processes mapped to the receiving physical process.

Two data structures (represented as sets) are utilized by the cloning mechanism to determine where a message is delivered, and whether additional cloning is necessary:

—`VSendSet`: the version numbers of the virtual processes mapped to the sending physical LP; and

—`VRcvSet`: the version numbers of the virtual processes mapped to the receiving physical LP.

The sender piggy-backs additional bits to the message so that the receiver can construct `VSendSet`. Before examining this process in more detail, we introduce the notation that will be used.

5.1 Notation

The original simulation is referred to as version 1 (the version number is incremented for each clone that is created). V denotes a virtual logical process and P denotes a physical logical process. V_A^i is virtual logical process A version i. Similarly, P_A^i refers to version i of physical logical process A. It can be shown that the version number of a physical logical process is the same as the *lowest* version of the virtual logical processes mapped to that physical LP.

A cloned simulation passes through three distinct phases: (1) before cloning (2) diffusion (before the simulation is fully replicated) and (3) fully replicated. Before a simulation is cloned there is one set of Vs and one set of Ps. At this stage V_j^1 maps to P_j^1 for all j. As an example, consider a simulation composed of three logical processes (before cloning):

—$V_A^1 \rightarrow P_A^1$
—$V_B^1 \rightarrow P_B^1$
—$V_C^1 \rightarrow P_C^1$

Where → means "maps to." This scenario is illustrated in Figure 4. The virtual logical processes are at the top, and the physical logical processes are at the bottom of the figure.

5.2 Decision Point

A simulation is cloned at a *decision point*. The decision point defines *when* and *how* the new version differs from the original version. The semantics of inserting a decision point (creating a new simulation) are (1) to define the set of virtual logical processes (referred to as the *clone set*) that immediately differ in the new version compared to the original, and (2) to define how the LPs in the clone set of the new version differ from the original version. The clone set is

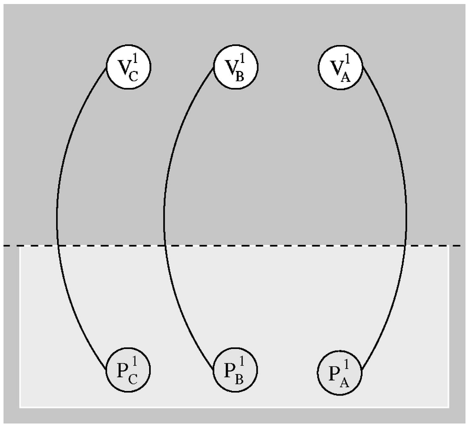

Fig. 4. A snapshot of a simulation consisting of three logical processes before the instantiation of a clone; the mapping of the virtual processes to physical processes A, B and C.

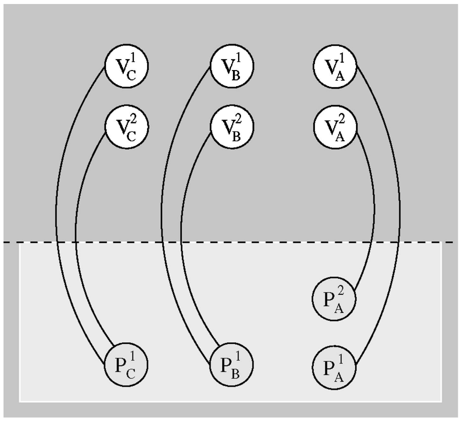

Fig. 5. A snapshot of a simulation that has been cloned; the mapping of the virtual processes to physical processes A, B and C.

denoted `CloneSet`^{i}, where i refers to the *new* cloned version. All virtual logical processes in a `CloneSet` must belong to the same version (i.e. the decision point cannot span multiple versions).

When the simulation is cloned, a new complete set of virtual processes is created. One set represents the original simulation and the other represents the new version. Cloning also creates a new set of physical logical processes; one physical logical process is created for each virtual logical process in the `CloneSet`. The state of these new physical logical processes differs from the original according to the definition of the new version of the simulation.

Immediately after the clone is created, V_j^1 maps to P_j^1 for all j, V_k^2 maps to P_k^2 for all k that are part of the clone set, and V_l^2 maps to P_l^1 for all other virtual LPs. Figure 5 shows the mapping between virtual and physical logical processes immediately after the simulation is cloned, where the clone set consists of a

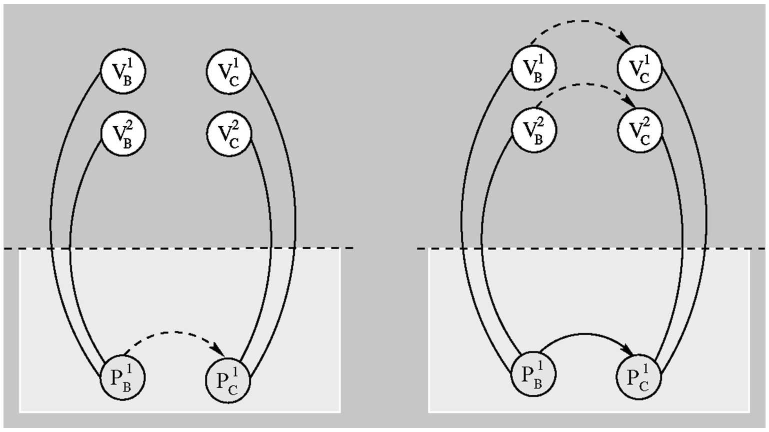

Fig. 6. Transparent send and receive between virtual logical processes.

single virtual LP, V_A^1. The mapping between virtual and physical processes for the new version of the simulation is as follows:

—$V_A^2 \rightarrow P_A^2$
—$V_B^2 \rightarrow P_B^1$
—$V_C^2 \rightarrow P_C^1$

Because V_A^1 is in the clone set and it maps to P_A^1, P_A^1 is cloned to create P_A^2. For virtual processes B and C, versions 1 and 2, share the same corresponding physical process.

5.3 Computational Sharing

Simulation computation and message sends and receives are carried out in the physical layer. A physical message send may correspond to a *set* of sends in the virtual layer. In the example (Figure 6), a message sent from physical process B to physical process C implements two virtual message sends from V_B^1 and V_B^2 to V_C^1 and V_C^2 respectively. Similarly, simulation computations in P_B^1 implement the computations performed by V_B^1 and V_B^2. In this way common computations among clones are only performed once.

5.4 Message Cloning

We have already seen that cloned versions start to diverge immediately after the clone set has been defined. As cloned versions of a parallel simulation progress, physical processes may need to be cloned, or messages originating from an un-cloned physical process may need to be forwarded (cloned). This section examines the process of determining whether or not a message must be cloned.

A message is cloned when the same physical sender is shared among two or more virtual senders, but the virtual receivers do not share the same physical receiver. To continue the above example, consider the case where process B now sends a message to process A (See Figure 7). Because both versions of the sender of the message are represented by the same physical process but there are two different physical processes representing the receiver, two copies of

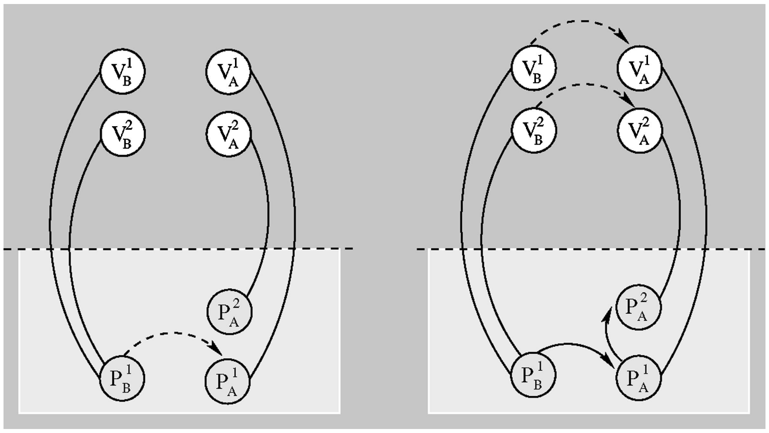

Fig. 7. Cloning a message.

the message must be sent, one to each physical receiver. The physical receiver determines this by inspecting `VSendSet` and `VRcvSet`.

In the example, when P_B^1 sends a message to P_A^1, `VSendSet` $= \{1, 2\}$, since V_B^1 and V_B^2 map to the sender P_B^1, and `VRcvSet` $= \{1\}$, since only V_A^1 maps to P_A^1. Each virtual process in the send set should have a corresponding virtual receiver. If the receiving physical process does not have a corresponding virtual receiver in the receive set then a copy of the message is forwarded to the appropriate physical logical process. In the example (See Figure 7), since version 2 is in the send set but not in the receive set, P_A^1 forwards the message to P_A^2.

The algorithm that determines which versions of the physical LP are to receive a cloned copy of the message is outlined below:

(1) get `VSendSet` from message
(2) get `VRcvSet` from local state
(3) $\texttt{forwardSet} = \texttt{VSendSet} \cap \overline{\texttt{VRcvSet}}$

5.5 Process Cloning

A physical logical process is cloned if (1) there is a virtual logical process in the receive set that should not be influenced by the incoming message or if (2) the destination address of the message is a physical logical process that has not yet been created.

Consider our running example of three logical processes. P_B^1 clones itself after receiving a message from P_A^1 (See Figure 8), since version 2 in the receive set should not be influenced by a message that is coming from a simulation that has been cloned (P_A^1 has cloned P_A^2). This corresponds to the sending of a physical message from P_A^1 to P_B^1. Because V_B^2 should not receive the message, version 2 of the virtual receiver should be prevented from being influenced by this message.

To implement this, the process is cloned *before* the event is processed so that the state of the physical process is not influenced by the incoming message. The new "version 2" of physical process B avoids the misconception that

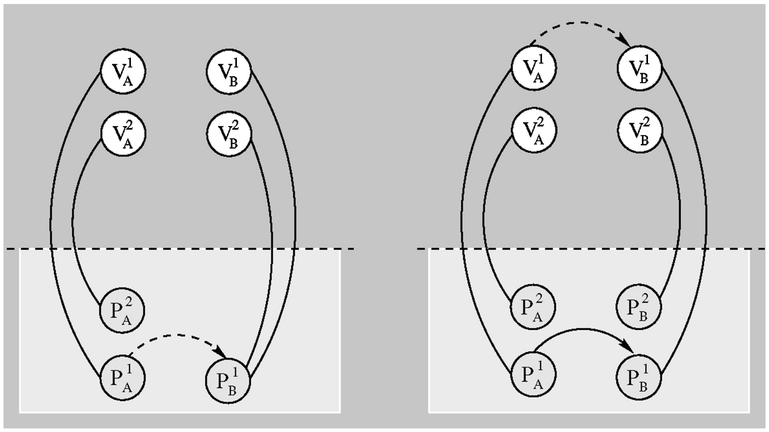

Fig. 8. Creation of a physical logical process (Case 1).

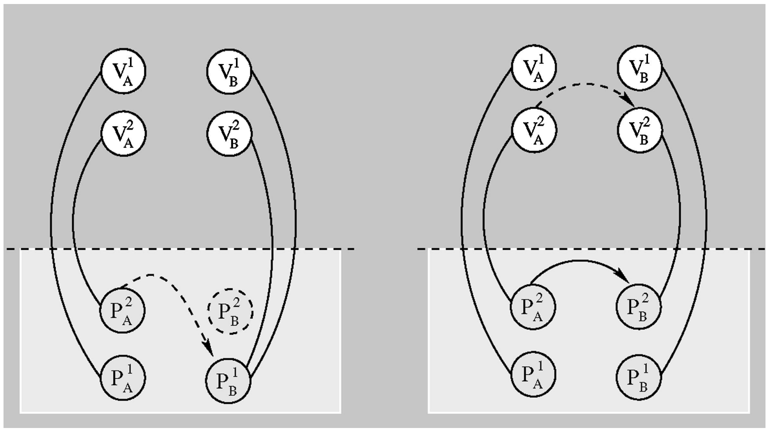

Fig. 9. Creation of a physical logical process (Case 2).

A processed the message. In this scenario the send set is $\{1\}$ because the set contains version 1 of virtual senders mapped to physical process P_A^1. The receive set is $\{1, 2\}$ since both version 1 and version 2 map to the physical receiver P_B^1.

Cloning occurs when the physical receiver maps to a virtual version not included in the virtual send set. Note that this case also includes the sending of a message to a receiver that has not yet been cloned (i.e. it is sent from a version number that is higher than the physical receiver). This case is illustrated in Figure 9. P_A^2 sends a message to P_B^2, but P_B^2 does not exist. So, P_B^2 is cloned from the process that contains the virtual logical processes mapped to the receiver (i.e. P_B^1).

Timely cloning of logical processes prevents receivers from being influenced by messages from versions not contained in the receive set. This is achieved by physically cloning processes and changing the mapping between virtual and physical processes. The physical logical process that receives the message computes a set called VPsMoveToClone to determine if it needs to be cloned. This set contains the virtual logical processes that should be prevented from being affected by the message. If the set is non-empty, a new physical logical process is cloned. The new physical process is assigned the same version number as the lowest virtual LP in VPsMoveToClone and these virtual LPs, are moved to the new cloned process. To accommodate the second case of cloning, a set called VPsRequested is defined. This set points to the virtual LPs that should be influenced by the message. If the version number of the physical receiver is *not* in this set, then these LPs are assigned to the VPsMoveToClone set. The algorithm used to determine process cloning is listed below:

```
get VSendSet from message
get VRcvSet from local state
VPsUnaffected = $\overline{\texttt{VSendSet}} \cap$ VRcvSet
VPsMoveToClone = VPsUnaffected
VPsRequested = VSendSet $\cap$ VRcvSet
if ( ! (PhysicalReceiver $\cap$ VPsRequested) ) then
    VPsMoveToClone = VPsRequested
if ( VPsMoveToClone <> $\emptyset$ ) then
    Cloned Physical LP $\leftarrow$ lowest( VPsMoveToClone )
```

5.6 The Four Cases of Cloning

To summarize, four outcomes are possible when a physical LP receives a message . The LP must be prepared to handle each of these four cases based on the mapping of the sender and receiver before receiving the message:

- **Computational Sharing**: The same physical sender and receiver are shared between virtual logical processes (e.g. Figure 6, virtual processes V_B^1 and V_B^2 both map to P_B^1 and both V_C^1 and V_C^2 maps to P_C^1). No message or process cloning is required.
- **Message Cloning**: The same physical sender is shared between virtual senders but the virtual receivers do not share the same physical receiver (e.g. Figure 7, V_B^1 and V_B^2 both map to the sender P_B^1, but V_A^1 and V_A^2 map to different physical processes). The message must be replicated and sent to both physical destinations.
- **Process Cloning**: The physical receiver maps to a virtual version that is not mapped by the sender (e.g. Figure 8, virtual receiver V_B^2, which is mapped to physical receiver P_B^1 should not see the message from virtual sender V_A^2). Note that this case also includes the sending of a message to a receiver that has not yet been cloned. The physical process that does not have a corresponding virtual sender must be cloned.
- **Replicated**: Virtual logical processes are independent of each other and do not share physical logical processes. No process or message cloning is required.

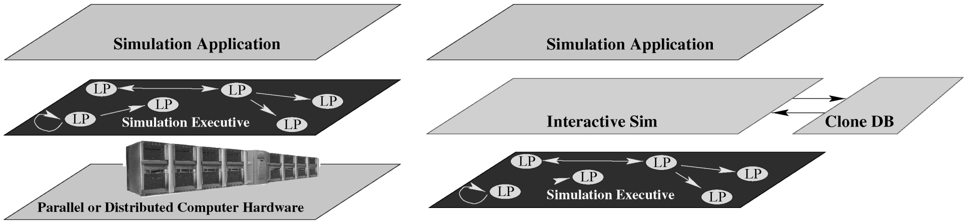

Fig. 10. Views of simulations: Traditional parallel discrete event simulation is shown on the left; the monitoring layer called interactive-sim in relation to the simulation executive and the simulation application is shown on the right.

Before proceeding to the discussion on the performance of the cloning algorithm we discuss how cloning is implemented.

6. CLONING IMPLEMENTATION

We define a simulation as being composed of a simulation application (provided by the user) that interacts with a simulation executive. The simulation executive provides primitives that allow simulation programmers to define their own applications and it implements the necessary underlying synchronization protocols. This is a layered software system, with the operating system at the bottom, the simulator executive in the middle and the simulation application at the top (See Figure 10).

Cloning is implemented in a software library named Clone-Sim that provides an application programmer interface (API) to the system. The library supports on-demand "cloning" of parallel and distributed discrete event simulations. Efficiency is accomplished by intercepting communication primitives (function calls) between the user application and the simulation executive. The approach will work for interactive as well as noninteractive environments and both optimistic and conservative simulators can be supported. The results reported here were accomplished with a Clone-Sim implementation using Georgia Tech's Time Warp simulation executive [Das et al. 1994] called GTW, an optimistic simulator.

The architecture of Clone-Sim is shown on the right of Figure 10. Interactive-Sim exists as a software layer between the simulation executive and the user's application simulation program. Interactive-Sim is decomposed into submodules, where each is implemented for a particular simulation executive. The submodules are "pluggable" in that the appropriate submodule is plugged in for a specified simulation executive. New submodules can be implemented using a specified application interface. The general idea behind Interactive-Sim is that it is transparent to the simulation program, as well as to the programmer utilizing the cloning primitives.

Clone-Sim is composed of two primary modules: **Interactive-Sim** which is layered between the simulation executive and the application simulation and the **Clone-DB** database. **Clone-DB** is independent of the synchronization primitives of the simulation executive. Interactive-Sim: (1) intercepts message sends and (2) processes events. Accordingly, Interactive-Sim emulates the simulation executive's message send and message receive function prototypes to intercept the invocation to process events or to forward copies of a message to cloned LPs. After interception, Interactive-Sim queries Clone-DB to determine whether message or process cloning is necessary (via respective inquire functions). Eventually Clone-Sim calls the simulation executive's actual message send and receive functions to accomplish the simulation. From the point of view of the simulation executive, Interactive-Sim is a simulator application. Details of the application programmer interface of cloning are beyond the scope of this paper, but are described elsewhere, please see Hybinette and Fujimoto [2002].

6.1 Assumptions

The relationship between the user application and the simulation executive is illustrated on the left of Figure 10. Here, the user application defines the events and the simulation executive handles synchronization issues. It is assumed that the simulation executive provides `ScheduleEvent` (send and schedule) and `ProcessEvent` (receive) primitives. If the application uses GTW as a simulation executive, it must define `ProcessEvent`, and tell GTW when to schedule the event. GTW then schedules a call to `ProcessEvent` at the appropriate time.

We also assume:

—that one can schedule events conservatively, that is, the event can be scheduled with the assumption that it will never roll back, (this is trivial if the simulation executive is conservative);

—the simulation executive supports dynamic LP creation and allocation, initialization and copying of LPs;

—simultaneous events are addressed by the underlying simulation engine (e.g. by prioritization).

Note that Clone-Sim ensures that simultaneous events do not interfere with cloning. This is done by prioritizing cloning events (such as instantiations of new simulations and physical processes) above all other events. The next section describes how cloning has been implemented using an optimistic simulator.

6.2 Implementation of the Optimistic Protocol

Clone-Sim has been implemented using GTW, Georgia Tech's Time Warp simulation executive [Das et al. 1994]. GTW is an optimistic simulator. Recall that events in an optimistic simulation may roll back. As mentioned above, the event that instantiates cloning must be a **conservative** event (guaranteed to never roll back). However, subsequent LP cloning events (within the same version of the simulation) may roll back.

A key challenge in the optimistic implementation is handling the rollback of events that may have triggered the cloning of a new physical LP. To address this, we use a special rollback function to keep LP mapping information consistent. When a message is received by an LP and the appropriate recipient does not yet exist, the parent accepts the event, clones itself, then sends a copy of the message to activate the new child LP. As a result, the original destination field in the event is overwritten with the parent's address. Before the destination is overwritten, however, it is saved in a separate field. During rollback, the original destination and mapping information is restored via the rollback function.

Two main data structures are used by the cloning mechanism, a relationship table and a mapping table. The relationship table keeps track of the shape of the version tree: which version cloned which other versions. This structure is modified when a simulation is cloned (not each time an LP is cloned). The data structure is read by LPs, upon the receipt of a message, to determine which other LPs are their children.

The mapping table is maintained by active LPs. The mapping table is part of the local state of the physical logical processes. This table is exclusive to the owner of the state. However, during instantiation of a clone, the parent initializes the index that belongs to its child. Thereafter, the child itself maintains its own information.

This mapping table contains (1) mapping information, (2) "birth" timestamps and (3) a bit indicating whether the corresponding LP is active or not. The mapping table is referenced to determine whom to clone and to whom messages should be forwarded. The `VRcvSet` discussed in Section 5 is constructed from the mapping table. It is also referenced to determine which LP to roll back or to determine which LP should handle a message when it is addressed to an LP that did not exist before the timestamp of the message.

7. CLONING EVALUATION

In order to establish the utility of the cloning mechanism, we evaluated the system in three ways. First, we presented performance results from a real world task and examined typical usage scenarios. Second, we looked at the performance improvement of cloning while controlling the rate at which physical LPs must be replicated. Finally, we examined the scalability of the approach both analytically and empirically.

Evaluation of cloning was conducted on two machines: an SGI Origin 2000 with sixteen 195 MHz MIPS R10,000 processors and on an SGI Power Challenge with twelve 75 MHz MIPS R8000 processors. On the Origin, the first level instruction and data caches are each 32 KB. The unified secondary cache is 4 MB. The main memory size is 4 GB. The first level instruction and data caches on the Power Challenge are each 16 KB, and the unified secondary cache is 4 MB. The main memory size of the Power Challenge is 3 GB. Each data point represents at least 5 runs for the Power Challenge and at least 15 runs for the Origin. The experiments use 4 processors on the Power Challenge unless otherwise indicated.

7.1 Air Traffic Control Simulation

To investigate the utility of cloning for real world tasks we implemented the cloning mechanism and examined typical usage scenarios for a commercial air traffic control simulation called the Detailed Policy Assessment Tool (DPAT). DPAT simulates air traffic patterns and computes congestion related delays and through-puts. The software can simulate both international airspace—Asia

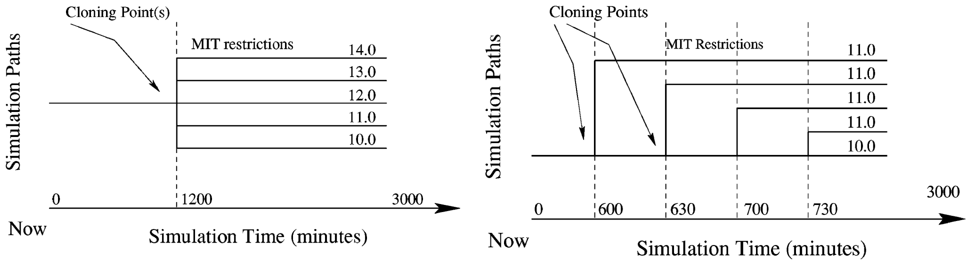

Fig. 11. DPAT usage scenarios. Left: several clones generated on several MIT values at the same simulation time. Right: one MIT value is evaluated at several initiation times. Times of the decision points are shown as dashed vertical lines.

and Taiwan—as well as the continental US (CONUS). For more detail on DPAT please see Wieland [1998].

The system models the behavior of each airport and the airspace between them. The airspace is divided into sectors. Each sector can enforce restrictions such as the number of aircraft occupying a sector at a specific time. In these experiments, two control restrictions affect the flow of aircraft: Ground-delay programs and miles-in-trail (MIT).

Departing aircraft can be regulated by ground-delay-programs. These restrictions prevent aircraft from departing an airport in order to avoid circling above the destination airport (which wastes fuel and can increase the risk of accidents).

En-route traffic is controlled by miles-in-trail restrictions that set a minimum distance between, aircraft. The allowed minimum distance differs, depending on weather conditions. In good weather the minimum distance allowed is typically 2 miles, while in bad weather (decreased visibility), the minimum distance is typically increased to 5 miles for greater safety.

The FAA can evaluate the effect of MIT restrictions and ground-delay-programs on traffic flow, using a simulation like DPAT. For example, an air traffic controller may wish to increase safety by increasing MIT restrictions provided this does not adversely affect the traffic flow rate.

Using dynamic cloning, we anticipate that more alternative MIT restrictions can be evaluated to manage the air space than if the simulation were simply replicated. To investigate this hypothesis, we evaluated typical usage scenarios of the DPAT software.

Two questions controllers often face regarding MIT restrictions are: *What* MIT value is the most beneficial and *when* should a new MIT value be enforced? These questions suggest two scenarios (depicted in Figure 11).

On the left, alternative values of MIT restrictions are imposed at the same time in the future. The figure depicts scenarios where a controller considers MIT values between 10 and 14. Each of the restrictions are activated 1,200 minutes from "now" (i.e. 20 hours from the current time). This point in time is labeled the "cloning point." We refer to this as the "forked" scenario.

On the right of Figure 11, different clones evaluate different points in time when a particular MIT restriction is imposed. Here, different simulations evaluate the same MIT restriction (11) enforced at various times in the future. For reference, the original MIT restriction of 10 is also evaluated. The time to enforce the restriction varies between 600 minutes from the current time to 730 minutes (the evaluations are initiated at 30 minute intervals). This is referred to as the "staggered" scenario.

The experiments using DPAT simulate 50 hours (3,000 minutes) of air traffic over the Continental US. This configuration consists of 738 sectors and 520 airports, or a total of 1258 logical processes. The total number of flights scheduled is 53,860. Typically, this scenario induces 349 delays of more than 15 minutes. Cloned simulations are instantiated on a sector near Chicago's O'Hare airport, sector "ZLC1N."

To stress the cloning mechanism, the experiments use an initial MIT restriction of 70. This is much larger than values typically used by the FAA, but larger numbers cause more congestion and thus lead to faster spreading of replicated LPs, making it a more challenging test case of the cloning software. Smaller MIT values (as those shown in the usage scenarios) lead to significantly better cloning performance. Cloning is evaluated by instantiating clones that increase the MIT value by a specific percentage. In the forked scenario each successive clone increases the value by 10%. In the staggered scenario each clone evaluates the same MIT value. The time difference between successive clones in the staggered scenario is 30 to 60 minutes.

DPAT is implemented on GTW (an optimistic parallel simulator) and is linked to the cloning library Clone-Sim. The DPAT source code includes 20,133 lines of C. Additional instructions are inserted into the sector code to detect preset cloning times and to identify the proper logical process (i.e. it ensures that ZLC1N induces cloning in the cloning set). At the proper time, a conservative event, schedules a cloning event, which in turn clones the process and sets the MIT value. The instructions added to DPAT to detect and initiate cloning were fewer than 100 lines of code.

7.2 Performance

We conducted a number of experiments to investigate the impact of dynamic cloning on DPAT simulation performance. For comparison, a batch-processing scheme, where each simulation path is run separately, one after the other, is compared with the dynamic cloning scheme. Performance is evaluated as speedup of the run time of cloning versus batch processing. Speedup is the total running time of the batch processing approach divided by the running time using dynamic cloning.

The independent variables of this experiment are the inception time of the cloning point (in simulated time) and the number of simulations. Both the "forked" and "staggered" scenarios are evaluated.

The plot on the left of Figure 12 shows the performance of DPAT on an Origin when several clones are generated on different MIT values at the same simulation time (the forked scenario). The plot on the right shows the performance of DPAT when one MIT value is evaluated at several different initiation times (for reference, the performance of a single simulation, where no clones are generated is shown as well).

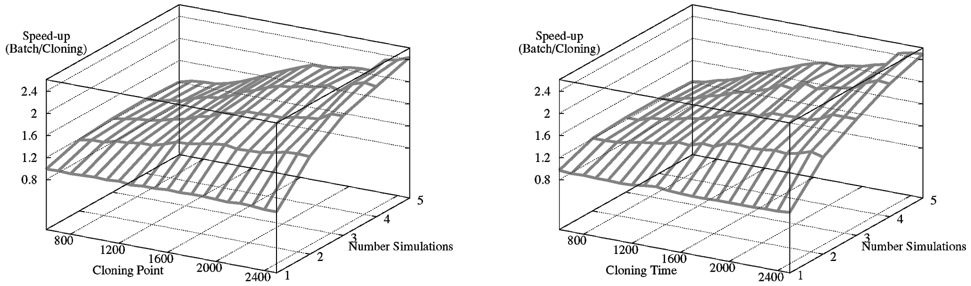

Fig. 12. Simulation cloning significantly improves the performance of DPAT over batch processing. Left: speedup of cloning over batch processing for DPAT when several clones are generated on different MIT values at the same simulation time. Right: speedup of DPAT when one MIT value is evaluated at several initiation times. Larger numbers indicate better performance.

The results indicate that dynamic cloning always outperforms the traditional batch scheme when running multiple simulations. As one would expect, simulation cloning becomes more advantageous as the number of alternatives increases or the later the decision point is inserted. The plots show that dynamic cloning runs up to 240 percent faster than the batch scheme (e.g. for simulating different MIT values: running 5 different MIT values at 2,500 simulated minutes takes 75.38 seconds for the dynamic scheme versus 183.60 seconds for the batch scheme).

7.3 Message Spreading Rate

The benefit of the incremental update scheme is evaluated by varying the rate at which one clone causes the creation of additional physical LPs. Our incremental update scheme avoids replicating the entire state space upon cloning the simulation. To quantify this effect we use a benchmark called P-Hold. P-Hold provides synthetic workloads using a fixed message population [Fujimoto 1990]. Parameters of the application allow us to control the spreading rate.

P-Hold works as follows: each LP is instantiated by an event. Upon instantiation, the LP schedules a new event with a specified time-stamp increment and destination. The destination LP is specified within the message.

To control the spreading rate of P-Hold, we adjusted the selection of the destination LP of a message (logical processes are instantiated by receiving messages). In the first case the spreading is slowed by having an LP send messages only to itself or to its neighbor (the local distribution). In the second case the spreading is not constrained and the destination is selected from a uniform distribution among all logical processes in the simulated system. The timestamp increment to schedule a new event is 1.0 in the local case and selected from an exponential distribution between 0.0 and 1.0 in the uniform case. In the cases of instantiated multiple simulations, each new clone is instantiated with the same time stamp. The experimental P-Hold simulations use a message population of 8096 and 1024 logical processes (for more detail on P-Hold please see Fujimoto [1990]).

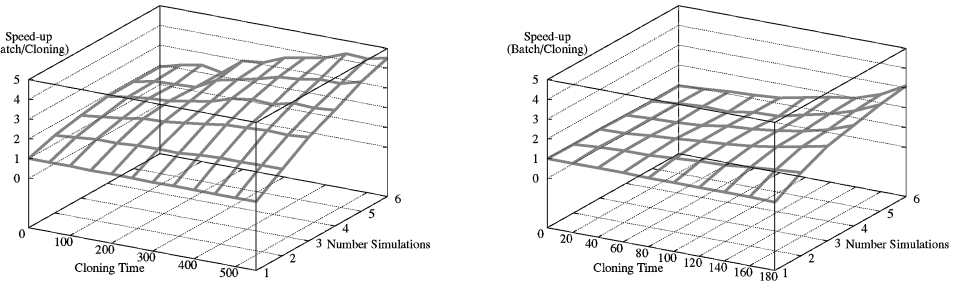

Fig. 13. Incremental updating improves the performance of P-Hold over batch processing. Several clones are generated at the same simulation time. Left: Speedup where destination LP is either self or nearest neighbor, the local case (slower spreading). Right: Speedup of cloning over batch processing where destination LP is selected from a uniform distribution of all LPs in the simulation (fast spreading). Larger numbers indicate better performance.

7.4 Performance

The independent variables for this experiment are the inception time of the cloning point (in simulated time) and the number of simulations. The cloning time varies between 0 and 600 simulated seconds in the local case and between 0 and 180 simulated seconds in the uniform case. There are zero to five new clones created for each case of P-Hold making it a total of 1 to 6 simulations run simultaneously.

The plot on the left of Figure 13 shows the performance of P-Hold in the local case versus batch processing. The plot on the right of Figure 13 shows the performance of P-Hold in the uniform case versus batch processing. When running multiple simulations, the results of a single simulation, where no clones are generated is shown for reference.

In the local case, dynamic cloning always outperforms the traditional batch scheme approach. In the uniform case however, the batch scheme performs better if cloning is initiated early and when evaluating more than 3 alternatives. This is because the spreading is quicker, and as a result very few computations are shared between the clones. If the cloning event occurs later, more computations are shared and again dynamic cloning outperforms batch processing.

As in the DPAT experiments, the performance advantage of cloning becomes more dramatic the later clones are spawned or the larger the number of alternatives. For the uniform case, dynamic cloning improves performance by as much as 313 percent and in the local case by as much as 450 percent (e.g. instantiating 6 clones at virtual time 550 takes 53.04 seconds for the dynamic scheme versus 239.27 seconds for the batch processing scheme). As one would expect, the plots indicate that cloning is more advantageous for simulations using local interactions.

The dramatic improvement over the batch-processing is due to the incremental updating scheme. The performance gain of the local case highlights that incremental updating further improves the benefit of dynamic cloning when the behavior of the cloned simulations diverges slowly.

8. SCALABILITY

We now examine the scalability of cloning. Specifically, we examine how cloning impacts the performance as we increase the *number* of clones. For a replicated simulation application with N execution paths, each LP is duplicated N times. In many cases, messages are duplicated N times as well. On average, we expect a replicated simulation to consume N times the resources (in terms of space and time) as a simulation with a single execution path.

In the best case our cloning approach is significantly more efficient than replication (as much as N times faster). In the worst case, cloning is only slightly less efficient than replication. In this section we show that this efficiency is *scalable* with respect to the number of clones. By that we mean that cloning maintains its advantage over replication regardless of the number of execution paths.

When evaluating the performance and scalability of a cloned simulation it is important to keep in mind that there are three key phases of an application's "life cycle" as follows:

- **Before decision point:** in this phase the simulation has not been cloned, and there is a single execution path. It is during this phase that cloning offers a tremendous advantage over replicated simulations. All events processed in this phase run once, whereas in the replicated case the events must be processed redundantly in each copy of the simulation.
- **Spreading:** our approach uses an incremental method that only duplicates (clones) LPs as necessary. At the decision point only a small number of LPs must be replicated. In this phase a large proportion of computations can be shared, and the approach retains a significant advantage over replicated simulations. However, as the replicated LPs interact with other LPs it becomes necessary to clone the affected LPs as well.
- **Spreading complete:** eventually all LPs will have been cloned. From this point on, the fully cloned simulation must process the same number of events as a replicated simulation. The additional overhead of cloning in this case is a single comparison per event. Cloned simulations in this phase are only slightly more expensive than replicated simulations.

The performance improvement of a cloned simulation in comparison to replicated simulations depends on the relative length of each of these phases. For instance, if the decision point occurs late in the simulation, the cloned simulation would have a tremendous performance advantage over replicated simulations. On the other hand, the worst case for cloned simulations is an early decision point combined with rapid spreading. In this case the cloned simulation's performance will be slightly worse than a replicated simulation (an extra comparison for every event).

Before proceeding to empirical results, we consider how the number of clones and the number of processing elements (PEs) may impact the scalability of the cloning algorithm.

8.1 Scalability: Number of Clones

Before the first decision point, every message applies to every possible clone and the work of processing a message is automatically shared between the future cloned execution paths. In comparison to a single copy of a replicated simulation, our cloning algorithm requires an additional operation to determine if it is in this phase. Because the results of events processed during this phase are automatically shared between the future clones, we realize up to N times higher performance with respect to replicated simulations.

After spreading is complete, each message applies to only *one* clone, so additional processing to determine whether it should be forwarded is not required. The overhead of cloning in this phase is just one comparison per received message—the LP must determine whether the simulation has reached the "fully replicated" phase of the simulation. Because of the additional comparison per event, performance of the cloning approach is slightly worse than replication in this phase of the simulation.

Analysis of performance during the spreading phase is a bit more complicated. As the number of clones is increased, performance is impacted during the spreading phase in three ways. As the number of clones increases: (1) each LP must maintain additional state information to keep track of the set of clones to which it belongs, (2) the amount of information appended to messages increases because the sender must communicate which clones it belongs to (see Section 5 for additional details), and (3) the time required for an LP to process each incoming message increases.

(1) and (2) are information storage, or *space* costs. Additional information is needed at the LPs and in the messages during the spreading phase to enable a receiving LP to compute whether it will need to clone itself or perhaps forward the message to a cloned version of itself.

We can estimate the storage requirements for (1) and (2) using information theory as follows. Assume there are N clones. In addition to the content of the message used by the simulation application, the sender must also communicate which subset of the N clones it belongs to; there are $2^N - 1$ possible subsets, or $2^N - 1$ possibilities for this part of the message. According to Shannon, if all of the approximately 2^N possible subsets are equally likely, the information content of this portion of the message is $\log(2^N - 1) \approx N$ bits [Shannon and Weaver 1949]. Essentially, one bit is required for each clone. This is the worst case. If we know *a priori* that certain subsets are more likely than others, advanced coding techniques may be employed to reduce this space overhead (e.g. Huffman coding, see for example Cover and Thomas [1991]).

In terms of information storage requirements, the overhead scales linearly with respect to the number of clones. In comparison, N replicated simulations would require the same message to be sent N times. In a cloned simulation we can send *one* message with N additional bits appended. We are, in effect, compressing each additional message into one bit.

During the spreading phase, the time to process a message once an LP receives it, increases linearly with respect to the number of clones. Processing a message involves at most: (1) four set intersections, (2) three set inverses, and (3) forwarding the message to the corresponding cloned LPs (see Section 5 for more details on the computation). It is possible on some hardware for (1) and (2) to be computed in *constant* time (otherwise, it is at worst $O(N/$ sizeof(largest bit vector on hardware))). If parallel multicast is available, (3) can be accomplished in constant time as well. Empirically, the cost for these operations is insignificant in comparison to the other costs of processing a received message. In other words, cloning overhead is much less expensive than duplicating the entire processing of a message as would occur in a replicated simulation.

To summarize scalability with respect to the number of clones: Before the first decision point, the overhead of cloning is minimal and performance in comparison to a replicated simulation is almost N times better. After spreading is complete, the overhead, in comparison to a fully replicated simulation, is one comparison operation per event—performance is slightly worse than replication. Finally, during the spreading phase, required resources (space and time) increase linearly with the number of clones but are always *less* than the cost of full replication.

8.2 Scalability: Number of PEs

For many parallel algorithms, as the number of PEs is increased, speedup becomes sub-linear because synchronization and communication overheads begin to dominate. Our cloning algorithm, however, only requires synchronization at one point: the initialization of a new clone. No other inter-processor synchronization is necessary. Therefore the cloning algorithm scales with the number of PEs to the same extent as the underlying parallel simulation. The cloning overhead does not increase as the number of processors increases, although of course applications may be subject to inefficiencies in the underlying simulation executive. Our implementation uses the GTW executive, which has been shown to be efficient empirically [Das et al. 1994].

8.3 Scalability: Performance

In this section we evaluate scalability experimentally with regard to (1) the performance as number of clones increase, (2) the performance improvement as more processors are available to run the simulation, (3) the impact on run time performance as the number of clones (and LPs) increase and (4) the impact on performance for a given number of clones as the message population increases.

To evaluate how the number of clones impacts performance, a suite of experiments was run using the P-Hold benchmark application. The number of clones was varied from 1 to 32, while the number of PEs was varied from 1 to 16. The number of LPs was varied from 4 to 1024, and the message population was varied from 64 to 8192. All simulations ran for 300 simulated seconds. For comparison, we also ran a replicated simulation under the same conditions. In these experiments, the decision point for cloning was set early (at 10 simulated seconds) in the simulation to create a worst-case situation for cloning.

A plot showing representative performance is presented in Figure 14. Performance is run-time excluding initialization and wrap-up. This plot depicts

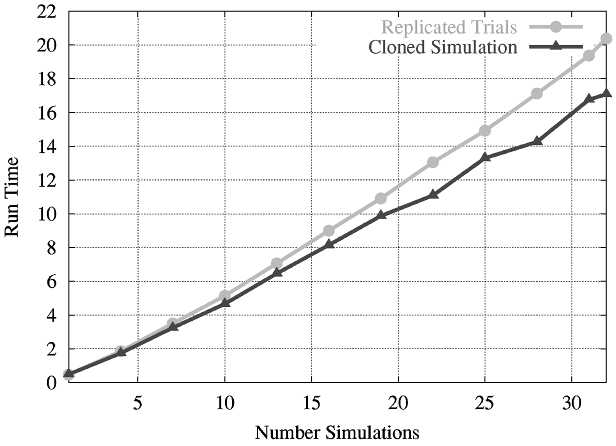

Fig. 14. Empirical performance of our method scales with the number of clones. Performance is measured as the run time of the simulation, excluding initialization and wrap-up. Smaller numbers indicate better performance.

a series of P-Hold runs on 4 processors using 256 logical processes and a message population of 128. Each data point is the average of 5 runs. This plot shows that as the number of clones increases, run time increases linearly. It also shows that in all cases, cloning is more efficient than replication. The advantages of cloning over replication scale with respect to the number of clones. Similar results using the DPAT and again the P-Hold application are provided in Section 7.2 and 7.4.

To assess potential sequential bottlenecks we examined the impact on performance as the number of processors and the number of clones is varied. Performance is measured as the run time of the simulation, excluding initialization and wrap-up (shorter run times indicate better performance).

The plot in Figure 15 illustrates the impact on run time as the number of processors is varied from 1 to 16 and the number of simulations is varied from 4 to 18. This simulations use 256 logical processes and a message population of 1024. All simulations run for 1000 simulated seconds and are cloned midway, at 500 simulated seconds. The run time and number of processor axes are plotted on a log scale. The planar shape of the resulting plotted data shows that speedup is linear with regard to the number of PEs.

8.4 Summary of Scalability Results

In this section we have argued that cloning is almost always more efficient than simulation replication. In some phases of a simulation, we can provide up to N times higher performance (where N is the number of execution paths).

The situation where replicated simulation has an advantage occurs when a decision point is placed early in simulated time and spreading is rapid. This forces the cloned system to fully replicate all LPs; the system must provide a separate execution path for every clone. However, in this worst case, cloning is

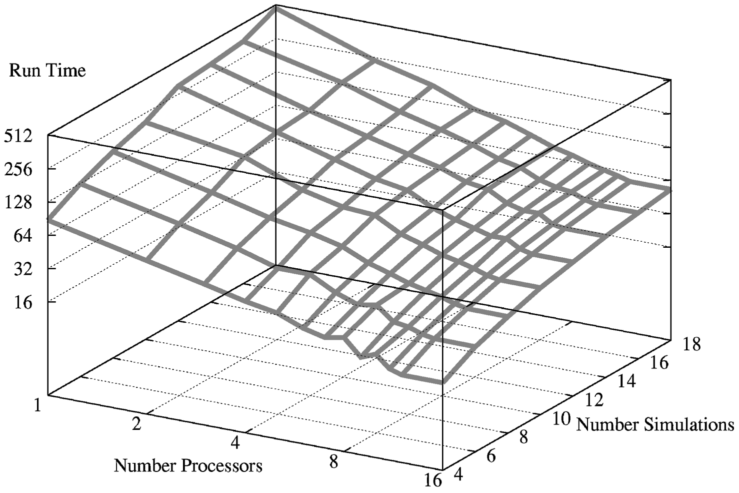

Fig. 15. Cloning scales with the number of processors. Performance is measured as the run time of the simulation, excluding initialization and wrap-up. Smaller numbers indicate better performance.

only slightly less efficient than replication; only one additional comparison is required to process each event.

Our analysis shows that the advantages of cloning are preserved regardless of the number of clones (or execution paths). This analysis is confirmed empirically through a number of experiments in which multiple parameters of the simulation are varied.

9. PRUNING EVALUATION

Just as it is advantageous to add clones to a running simulation, it may also be useful to enable a user to terminate a simulation that is no longer needed. This releases computational resources so they can be used by the remaining clones. Simulation pruning may be interactive or activated via a trigger function defined at the insertion of the cloning point.

Performance improvements due to pruning can be observed by measuring the progress of virtual time as a function of real time. In these experiments, 2 days of air traffic delays or 3000 simulated minutes are evaluated. Cloning occurs at simulated time 804 and pruning is initiated by a trigger that is installed at the insertion of the decision point. The trigger is set to be activated at simulated time 1,116. The trigger then checks if subsequent events will activate a cloning point. The trigger is a conditional statement that inserts a decision point when executing an event that has a time stamp later than 1,116. In these experiments an event with a time stamp of 1166.70 inserts a cloning point at the current simulated time (i.e. the cloning point is inserted at 1166.70).

The plot on the right of Figure 16 shows three curves: the curve on the left depicts a single simulation shown for reference, the middle and right curves show runs of two simulations each, where cloning occurs at simulated time 804. The simulation depicted by the the right curve is never pruned while the clone in the simulation plotted in the middle curve is pruned at simulated time 1,116.70.

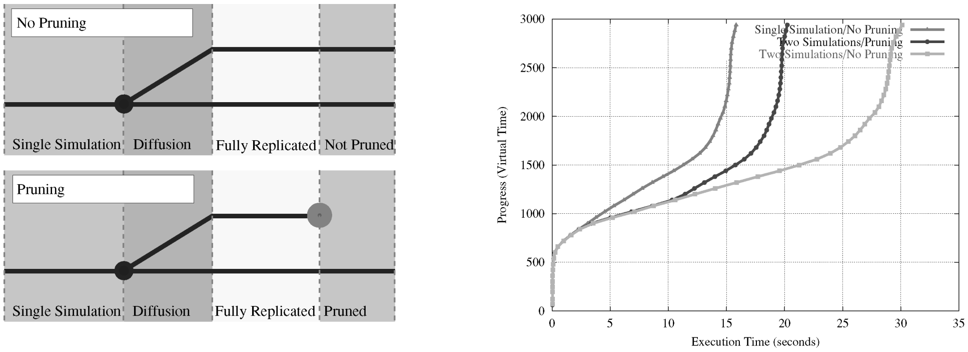

Fig. 16. Pruning of DPAT.

The acceleration of simulated time due to pruning is noticeable at the activation of the pruning trigger as a substantially increased slope (at simulated time 1,116.70). The pruned curve immediately conforms to the slope of the single simulation curve.

An estimate of the performance of a cloned simulation that is pruned can be estimated by factoring the single simulation execution time into the proportion of time spent at the different phases of cloning:

$$\begin{aligned}\text{Estimated Time} = {} & \text{single simulation proportion} * \text{single time}\\ & + \text{diffusion simulation proportion}\\ & \quad * \text{diffusion savings factor} * \text{single time}\\ & + \text{replicated simulations proportion before pruning}\\ & \quad * \text{\# prunes} * \text{single time}\\ & + \text{replicated simulations proportion after pruning}\\ & \quad * \text{\# prunes} * \text{single time}\end{aligned}$$

The “diffusion savings factor” depends on how much of the processes’ state is shared between clones. In this estimate, pruning is assumed to occur after diffusion[1] (see illustration on left in Figure 16). Since a pruned simulation, is compared with an unpruned simulation, the first three factors are eliminated in the subtraction.

As an example, consider the performance curves described above. Here, the simulation completes at simulated time 3,000. Each run completes in real time at 15.93 seconds for the single simulation run, 20.33 seconds for the cloned and pruned run, and 30.30 seconds for the cloned run (note that pruning improves the performance by 33 percent over not pruning the simulation). In this example the pruning heuristic predicts that the performance improvement should be 9.73 seconds:

$$\begin{aligned}\text{Estimate Time Benefit} &= \frac{3,000 - 1166.70}{3,000} * 15.93\\ &= \mathbf{9.73}\ \text{seconds}\end{aligned}$$

[1] Diffusion is the spreading phase of cloning; this is before the simulation is fully replicated.

The actual performance benefit is:

$$\begin{aligned}\text{Actual Time Benefit} &= 30.30 - 20.33 \\ &= \mathbf{9.97}\ \text{seconds}\end{aligned}$$

In this case the estimate differs 2.4% from the actual performance. The above methodology can be used for multiple prunes with the same pruning time. For clones with different pruning time, the heuristics may be applied to each clone separately then adding the time benefit of each pruned simulation.

Note that the implementation allows pruning to occur during the diffusion phase, even though our evaluation heuristic assumes otherwise.

10. CONCLUSIONS AND FUTURE WORK

Simulation cloning enables the exploration of multiple possible futures in interactive parallel simulation environments. The mechanism may be applied to simulations using either conservative or optimistic synchronization protocols.

Simulation cloning improves simulation performance in two ways: First, simulations that are interactively cloned at a decision point are able to share the same execution path before the decision point, so the computations before the decision point only need to be executed once. Second, after cloning, multiple versions of the simulation are able to share computations and messages via a construct called *virtual logical processes*. Additionally, clones can be created "on-the fly" and "pruned" when it can be determined that they will not provide useful results.

The benefit of execution path sharing is demonstrated by evaluating a commercial air traffic control simulation. Our performance results show that cloning can significantly reduce the time required to compute multiple alternate futures. These results also show that dynamic cloning becomes more advantageous as the number of alternatives increases or the later the decision point is inserted.

We also demonstrate that virtual logical processes are especially efficient when the influence of a message introduced by a logical process does not spread to other processes in the simulation at a high rate. This is shown by using a benchmark that varies the rate of cloning other LPs. The advantages of cloning are preserved regardless of the number of clones (or execution paths).

Efficient implementation of the cloning mechanism is achieved by intercepting the communication primitives of the simulator executive. By monitoring send and receive primitives, cloning can determine when to clone a logical process and when to forward a message (clone a message). To accomplish this, the LPs maintain a data structure used to determine mappings to virtual logical processes. Additional information is piggy-backed by the sender on messages so the receiver can determine whether to clone a process or to forward a message.

Our work focuses on eliminating inefficiencies arising from the use of replicated trials for testing the impact of different policies on performance (as might be the case in an air traffic control application). However, researchers often use replicated trials with different random number seeds to determine statistical measures such as the mean and confidence interval of a performance variable.

The best use of cloning in stochastic simulation remains an open question. Because if the random numbers differ at the start (by far the most common scenario), the replications may have little computation in common. If this is the case, simulation cloning will not be advantageous. However, one can use cloning in conjunction with traditional replication, the cloning being used to evaluate alternate parameter settings, and replicate a *set* of clones to address the random number issue.

Consider an example where a researcher would like to test five different policies and that he would like to test each one five times (for statistical significance). Suppose, also, that the different policies are applied mid-way through the simulation. Ordinarily one would have to run 25 replicated simulations to test each of the five policies five times. Cloning can be employed in this example as follows. The simulation is cloned five times at the start; each clone is initialized with a different random number seed. Mid-way through the simulation, each clone is cloned five times (one for each policy). The final result is 25 different execution paths—each policy is tested five times with different random number seeds. This approach is efficient because computations can be shared at each stage of the simulation.

Currently, it is assumed that a user can interject decision points into the simulator. Defining a mechanism for automating the interjection of decision points is a topic for future research. This may be accomplished by balancing the “running ahead” and “branching” depending on the probability of particular branches against available computational resources and real time constraints.

ACKNOWLEDGMENTS

The D-PAT application was written by Frederick Wieland of the MITRE Corporation and we are also grateful for his helpful comments on this work.

REFERENCES

BESTAVROS, A. 1994. Multi-version speculative concurrency control with delayed commit. In *Proceedings of the 1994 International Conference on Computers and their Applications*.

COVER, T. AND THOMAS, J. A. 1991. *Elements of Information Theory*. Wiley Series in Telecommunications. John Wiley & Sons.

DAS, S., FUJIMOTO, R., PANESAR, K., ALLISON, D., AND HYBINETTE, M. 1994. GTW: A time warp system for shared memory multiprocessors. In *Proceedings of the 1994 Winter Simulation Conference Proceedings*. 1332–1339.

DAVIS, W. J. 1998. On-line simulation: Need and evolving research requirements. In *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, J. Banks, Ed. John Wiley & Sons. Co-published by Engineering and Management Press.

FRANKS, S., GOMES, F., UNGER, B., AND CLEARY, J. 1997. State saving for interactive optimistic simulation. In *Proceedings of the 11th Workshop on Parallel and Distributed Simulation (PADS-97)*. 72–79.

FUJIMOTO, R. M. 1990. Performance of time warp under synthetic workloads. In *Proceedings of the SCS Multiconference on Distributed Simulation*. Vol. 22. SCS Simulation Series, 23–28.

GLYNN, P. AND HEIDELBERGER, P. 1991. Analysis of parallel replicated simulations under a completion time constraint. *ACM Trans. Model. Comput. Simul. 1*, 3–23.

GOLDBERG, A. P. 1992. Virtual time synchronization of replicated processes. In *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS-92)*. Vol. 24. SCS Simulation Series, 107–116.

HEIDELBERGER, P. 1991. Discrete event simulations and parallel processing: Statistical properties. *SIAM J. Sci. Statis. Comput. 9*, 1114–1132.

HENRIKSEN, J. O. 1997. An introduction to SLX. In *Proceedings of the 1997 Winter Simulation Conference*. 559–566.

HYBINETTE, M. AND FUJIMOTO, R. M. 2002. Scalability of parallel simulation cloning. In *Proceedings of the 35th Annual Simulation Symposium (SS-2002)*. IEEE Computer Society Press.

LENTZ, K. P., MANOLAKOS, E. S., CZECK, E., AND HELLER, J. 1999. Multiple experiment environments for testing. *J. Elect. Test.: Theory Appl. 11*, 3 (December), 247–262.

SCHNEIDER, F. 1990. Implementing fault-tolerance services using the state machine approach: A tutorial. *ACM Comput. Surveys 22*, 4, 299–320.

SCHRIBER, T. J. AND BRUNNER, D. T. 1997. Inside discrete-event simulation software: How it works and why it matters. In *Proceedings of the 1997 Winter Simulation Conference*. 14–22.

SHANNON, C. E. AND WEAVER, W. W. 1949. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, IL.

STEINMAN, J. 1991. SPEEDES: Synchronous parallel environment for emulation and discrete event simulation. In *Advances in Parallel and Distributed Simulation*. Vol. 23. SCS Simulation Series, 95–103.

VAKILI, P. 1992. Massively parallel and distributed simulation of a class of discrete event systems: A different perspective. *ACM Trans. Model. Comput. Simul. 2*, 3, 214–238.

VON NEUMANN, J. 1956. *Probabilistic Logics and the Synthesis of Reliable Organism from Unreliable Components*. Princeton University Press.

WIELAND, F. 1998. Parallel simulation for aviation applications. In *Proceedings of the IEEE Winter Simulation Conference*. 1191–1198.

ZAYAS, E. 1987. Attacking the process migration bottleneck. In *Proceedings of the 11th ACM Symposium on Operating System Principles*. 13–24.

Received April 2000; revised March 2001 and January 2002; accepted February 2002