Dynamic Virtual Logical Processes

Maria Hybinette Richard Fujimoto

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Abstract

The implementation of a cloning mechanism that allows for the evaluation of multiple simulated futures is presented and its performance is analyzed. 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 in gaming, tactical and strategic applications can be evaluated interactively or non-interactively. Performance results show that virtual logical processes, a new mechanism developed to avoid repeating common computations among clones improves efficiency.

1 Introduction

The goal of the research described here is to provide a decision aid that allows engineers and scientists to simulate complex situations and facilitate rapid and informed planning. Different possible futures are explored and compared by cloning an on-going parallel simulation. The tool supports critical decision making such as determining whether or not to mobilize reinforcements in a military scenario. Other applications that may benefit include gaming, strategic and tactical battle management and air-traffic scheduling.

A powerful simulation-based decision tool should provide for pausing, steering, monitoring, rollback and the evaluation of alternatives (a generalization of what-if scenario analysis). Pausing enables participants to interject new execution paths or to modify variables of the current execution path. Execution path modifications are added to the simulation as *decision points*. Steering is the insertion of decision points without pausing. Monitoring provides for querying and sampling simulation variables. These variables may be sampled continuously, triggered when some pre-defined condition is satisfied or presented on demand. Rollback provides for the re-evaluation of old decision points and variables. While the design and implementation support all these features, this paper focuses on the details of the parallel evaluation of multiple alternatives.

Related work in interactive parallel discrete event simulation is described in [11] and in [4]. In [11] 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 [4] allows for the testing of what-if scenarios provided they are interjected before a deadline. Alternatives are examined one after the other and the simulation must undo the effect of the previous alternative before considering another. In contrast to these methods, our algorithm dynamically creates and evaluates multiple alternatives concurrently using a cloning mechanism. New versions of a simulation in progress are cloned to evaluate alternatives. The alternative simulation proceeds in parallel with the original. Thus an increasing number of alternatives can be evaluated before resolution.

Parallel discrete event simulators provide speed up for time-consuming applications such as strategic battle-management, telecommunication networks, air-traffic scheduling and other large scale simulations. Strategic and tactical battlefield simulation in particular would benefit from our mechanism, since they often weigh options and must be responsive to unanticipated developments during a scenario.

The interactive system is intended for parallel discrete event simulators, where objects in the simulation are emulated by logical processes (LPs). The simulation progresses as LPs exchange time-stamped event messages that model changes of the system state at discrete points in time. In order to maintain consistency, events are processed in time-stamp order according to a consistency protocol. The two prevailing consistency protocols are *conservative* and *optimistic*. The conservative protocol enforces consistency by avoiding the possibility of ever receiving an event in the past. The optimistic protocol, in contrast, provides for receiving an event from the past by rolling back previously processed events with earlier time-stamps than the one received. Our cloning mechanism is appropriate for both conservative and optimistic simulation protocols.

Cloning was suggested by von Neumann [12] more than 40 years ago to provide fault-tolerance. Fault-tolerance is ensured by providing identical processes, identical transactions, duplicate data, or redundant services [10]. Cloning can also improve throughput by placing process replicas in the proximity of where a service is needed [5, 10]. Cloning is also suggested as a solution for concurrency control in real-time databases [1] and to improve accuracy of simulation results [6, 9]. In the latter, the approach is to run multiple independent replications then average their results at the end of the runs.

The incremental update schemes of process migration algorithms such as [14] 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 in that only one active clone is provided for while we provide for multiple clones.

In this paper we describe the implementation and performance of a simulation system that provides for the exploration of what-if and alternative scenarios. The goal of this research is to develop a model that supports an efficient, simple, and effective way to explore and compare alternate scenarios. The initial design of the cloning mechanism is described in [7]. Here we extend this work by describing the implementation and demonstrating its performance.

The paper is organized as follows: The virtual logical paradigm is described in Section 2. This is followed by a discussion of the cloning mechanism in Section 3. We then describe the cloning mechanism implementation in Section 4. Next, performance measurements are presented and analyzed. We conclude with a summary and a discussion of future directions.

2 Virtual Logical Processes

A simple approach to provide for the exploration of computing multiple futures simultaneously is to replicate an entire simulation at a *decision point*, then independently execute each clone. Here, a decision point represents the location in the execution path where the state of the two versions start to diverge. One way to visualize this is to imagine the whole simulation domain replicated into many different planes where each plane is an independent version of the simulation executing in parallel with the other cloned versions (See Figure 1). However, this may not be suitable for large simulations because the cost of instantiating many objects (LPs) is prohibitively large. Furthermore, as the simulation progresses many computations may be duplicated between the planes even though they could be shared.

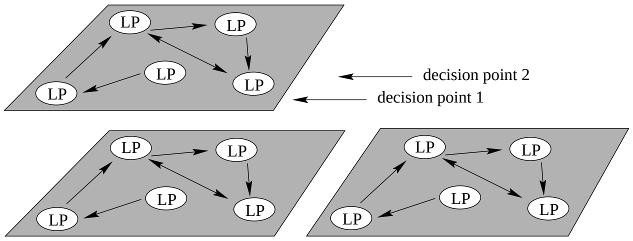

Figure 1: A Parallel Discrete Event Simulation Represented by Logical Processes (LPs) Replicated Twice; the Upper Left Plane Shows the Original Simulation

To reduce the cost of computing multiple-futures the state space of each cloned plane is updated incrementally via the construct of *virtual logical processes*. In this paradigm each plane or *version* of the simulation contains a collection of virtual logical processes. A collection of virtual logical processes is created each time a new version of the simulation is cloned. The difference between the cloned simulations is in the mapping of virtual logical processes to physical logical processes. Here, a physical logical process refers to the run-time environment of virtual logical processes. Each virtual logical process (V) is assigned or *mapped* to a physical logical process (P). The idea is that virtual logical processes can share the same physical logical process thereby avoiding replication of common computations. But two physical processes cannot be mapped to the same virtual process. An analog is virtual memory where the same main memory address can shared by several virtual addresses, but two addresses in main memory cannot be mapped to the same virtual address. The mapping between virtual and physical processes is updated as the clones diverge. Resources are re-used as long as possible and only the smaller portions which cannot be shared are replicated.

In the remainder of the paper the following notation is used:

- *version* i, the cloned simulation i.
- V, a virtual logical process.
- P, a physical logical process.
- V_j^i, the virtual logical process j version i.
- P_j^i, the physical logical process j version i.

The original simulation is version 1, the version number is incremented as simulations are cloned. The version number of a physical logical process is the same as the *lowest* version number of the virtual logical processes that maps to it.

Before a simulation is cloned there is one set of Vs and one set of Ps. A sample mapping of a simulation consisting of three logical processes before cloning is:

– V_A^1 maps to P_A^1

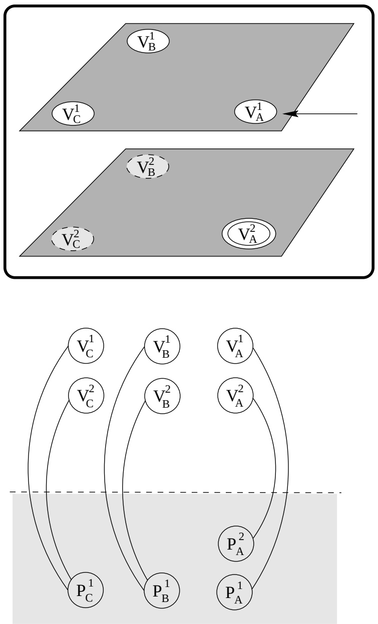

Figure 2: A Snapshot of a Simulation That Has Been Cloned; the Top Image Shows the Two Virtual Versions of the Simulation, the Bottom Image Shows the Mapping of the Virtual Processes to Physical Processes A, B and C

- V_B^1 maps to P_B^1
- V_C^1 maps to P_C^1

After a simulation is cloned a set of new physical processes are created for portions of the state that differ from the original simulation. The semantics of cloning is to instantiate the cloning *on* a set of physical logical processes called the clone set. The clone set may be specified dynamically. This represents the insertion of a decision point and is where the state of the versions start to diverge. In addition to the new physical processes, a new set of virtual processes is created, resulting in a total of two sets of virtual logical processes. The first set or version of virtual processes are the same as the the original simulation before cloning, the second set represents a new version of the simulation that differs in its mapping to physical logical processes. The mapping of the new virtual logical processes created from cloning the simulation of three logical processes described above *on* process P_A^1 is:

- V_A^2 maps to P_A^2
- V_B^2 maps to P_B^1
- V_C^2 maps to P_C^1

The bottom image in Figure 2 shows the mapping between virtual processes and physical logical processes after the simulation is cloned on physical process A. Here the mapping of the original version of virtual processes stays the same and the computation between clones is shared. Virtual processes B and C, version one and version two share the same corresponding physical process; while virtual process A version one and version two maps to different physical processes. As the simulation progresses the mapping of virtual processes to physical processes changes, as new physical processes are created. Message sends and receives are carried out in the physical layer. In this manner a physical send corresponds to a *set* of sends in the virtual process layer. In the example (see Figure 3), a message sent from physical process B to physical

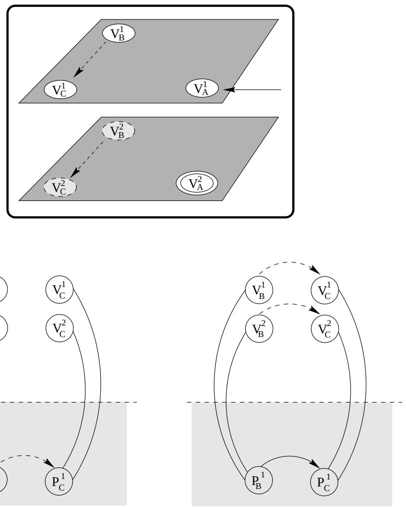

Figure 3: Transparent Send and Receive between Virtual Logical Processes

process C implements two virtual message sends from virtual process B version 1 and version 2 to virtual process C versions 1 and 2 respectively.

3 The Cloning Mechanism

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. As mentioned earlier, a parallel discrete event simulator is effected by the communication of messages. Thus, the reception of a message is the point where a logical process considers message forwarding and process cloning. Here, two sets are inspected: the virtual processes mapped to the physical send process `VSendSet` and the physical receive process `VRcvSet`. The sender adds information to the message so that the receiver can construct the send set.

To illustrate the approach consider a military tactical simulation consisting of three objects: A theater command center represented by logical process B, a tactical wing represented by logical process C and a platoon represented by logical process A. The platoon in the original simulation (version 1) has half manpower while the cloned platoon (version 2) has full manpower.

3.1 Message Cloning

A physical logical process first determines if the receiver message must be cloned. A message is cloned when an un-cloned physical process sends a message to a physical process that has been cloned. In the context of the military scenario this happens when the theater command center (B) requests a status report from the platoon (A). Since, both versions of the command center are represented by the same physical process but there are two different physical processes that represent the platoon – two copies of the message are sent (See Figure 4). This can be determined by inspecting the send set and the receive set. The send set is $\{1, 2\}$ and the receive set is $\{1\}$, because virtual logical process version 1 and 2 map to the senders physical process and virtual logical process version 1 maps to the receiver physical process. Each physical process that maps to a version that is in the virtual send set receives a message. Since version 2 is in the send set and not in the receive set the physical process that received the message forwards the message to physical process version 2. A process forwards a message to the lowest version that is not in the receive set. When a process forwards a message, the virtual processes that have seen the message (`MsgSeenSet`) is passed along as information in the message. This avoids redundant message sends. An algorithm that accomplishes these steps is listed below:

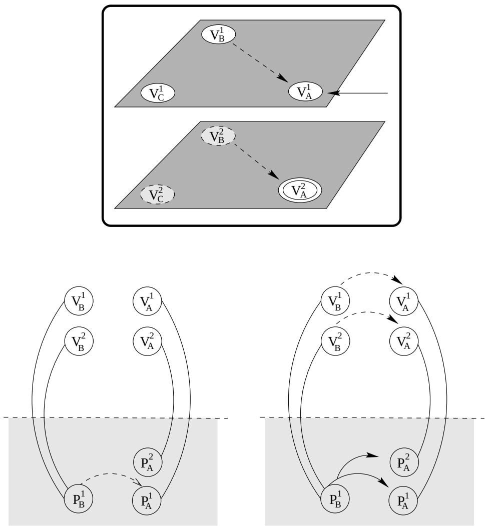

Figure 4: Multi-Casting a Message

1. get `VSendSet` from message
2. get `VRcvSet` from local state
3. `forwardSet` = `VSendSet` $\cap$ $\overline{\texttt{VRcvSet}}$
4. `forwardSet` = `forwardSet` $\cap$ $\overline{\texttt{MsgSeenSet}}$
5. `MsgSeenSet` = `MsgSeenSet` $\cup$ `VRcvSet`
6. forward message to the virtual logical process with the lowest version number in the `forwardSet`

3.2 Process Cloning

After determining the physical processes that need to receive a message, the generation of a new physical process is considered. A physical process is cloned if there is a virtual logical process in the receive set that does not have a corresponding virtual sender or if the destination address of the message is a physical logical process that has not yet been created. For example, the process representing the theater command center (P_B^1) clones itself in the military example if the half-staffed platoon (P_A^1) sends it a message to request extra manpower (See Figure 5). This corresponds to the sending of a physical message from P_A^1 to P_B^1. Since virtual process version 2 of processes B should not receive the message, version 2 of the virtual receivers should be prevented from being influenced by this message. This is achieved by changing the mapping between virtual and physical processes. The result is that the process is cloned *before* an 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 of B is prevented from the misconception that A needs extra manpower. 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 two maps to the physical

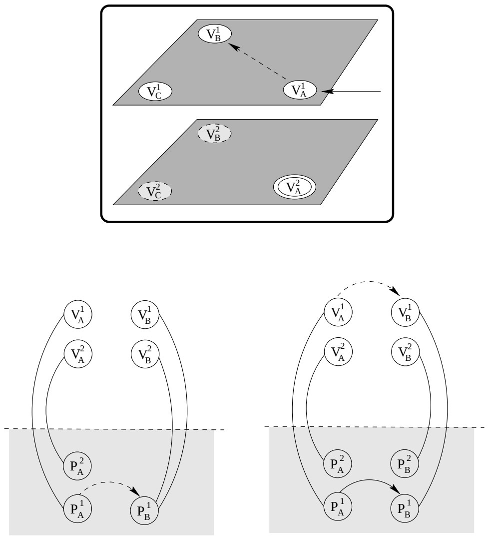

Figure 5: Creation of a Physical Logical Process

receiver P_B^1. The algorithm that determines process cloning is listed below:

1. get `VSendSet` from message
2. get `VRcvSet` from local state
3. `VPsUnaffected` = $\overline{\texttt{VSendSet}} \cap$ `VRcvSet`
4. **if** (`VPsUnaffected` <> $\emptyset$) then

   Clone a Physical LP and assign it the same version number as the lowest virtual LP in the set `VPsUnaffected`

4 The Optimistic Protocol

The optimistic implementation uses a rollback function to keep LP mapping information consistent. One implication is that if a recipient does not exist, the parent accepts the event, then clones the LP and then sends an instantiation message to activate a child. 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 within the message. During rollback the original destination and mapping information is restored via the rollback function.

There are two main data structures used for this procedure, a relationship table and a mapping table. The relationship table keeps track of the shape of the planes i.e., which planes cloned which other planes. This structure is modified when a simulation is cloned (not when a LP is cloned). The number of instantiated planes is only limited by memory. The data structure is read by other LPs to determine its children upon the receipt of a message.

The second data structure, the mapping table, is maintained by the active LPs. During instantiation of a clone the parent initializes the index that belongs to its child. Thereafter, the child itself maintains its own information. This data structure contains the mapping information, birth time stamps and whether the LP is active or not. To minimize mapping updates at the interjection of a decision point an LP keep track of which children are physically cloned. The physical LP can then compute its mapped virtual LPs. A sending LP piggy-backs this on a message so the receiver can compare its mapping to determine cloning or forwarding. The mapping table is referenced to determine whom to clone and to whom to forward messages. It is also referenced to determine which LP to rollback or to determine which LP should handle the message when a message is addressed to an LP that does not exist before the time stamp of the message.

5 Performance

Evaluation of the system was conducted on an SGI Power Challenge with twelve 75 MHz MIPS R8000 processors. The first level instruction and data caches are each 16 KB. The unified secondary cache is 4 MB. The main memory size of the power challenge is 3 GB. All experiments used four processors. The data collected for each plot are typical values of a series of at least 15 runs. Two benchmarks are used to measure performance: P-Hold and PCS.

P-Hold provides synthetic workloads using a fixed message population. 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 as well. The P-Hold simulations use a message population of 8096 and 1024 logical processes (for more details on P-Hold see [3]).

The second set of experiments use PCS, a benchmark which simulates a personal communication services network. The network is wireless (radio receivers and transmitters) and provides services to mobile PCS subscribers. The radio ports are structured in a grid with one port per sector. Each cell services *portables* or mobile phone units that each occupy a cell for a period of time before proceeding to one of the four neighboring cells. Call arrival, call completion, and move are example behaviors or types of portables which are modeled by events in the simulation. Cells are modeled by LPs. Here we use a PCS network of 1024 cells (a $32X32$ grid) and 8192 portables. Most of the communication, typically 90%, is between LPs residing on the same processor. Many of those messages are self-initiating, or sent between the same logical process. More details of PCS are described in [2].

To illustrate the impact of incremental cloning, the performance of replication (two physically replicated simulations) and a single cloned simulation are compared with the incremental cloning scheme. The replicated simulation immediately physically replicates all logical processes. The single clone simulation is a lower bound of cloning (one clone implies that the original simulation has not been cloned). In all cases the decision point occurs at one logical process (LP_0) at simulated time 100. Simulated time is also referred as the virtual time to distinguish it from wall-clock time.

Cloning is evaluated with P-Hold by varying the rate at which a clone causes the cloning of other LPs. This is done by adjusting 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 of all logical processes in the simulated system. The time stamp 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.

Figure 6 shows simulation progress (virtual time) of P-Hold in the local case versus wall clock time. Figure 7 shows the performance of P-Hold where the

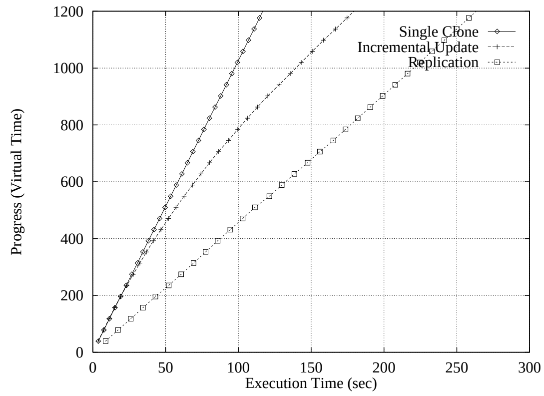

Figure 6: Performance of P-Hold Where Destination LP Is Either Self or Nearest Neighbor. Larger Numbers Indicate Better Performance

destination address is selected from a uniform distribution. In both plots a larger number indicates better performance.

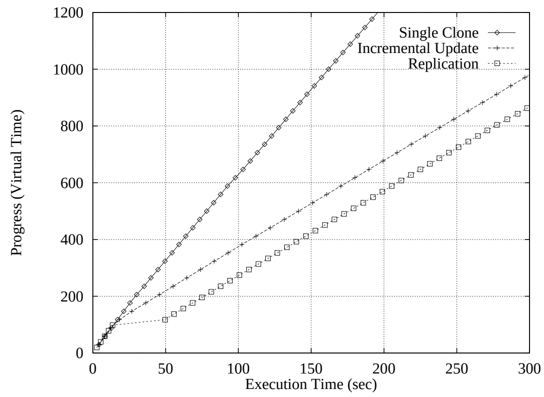

Figure 7: Performance of P-Hold Where Destination LP Is Selected from a Uniform Distribution of All LPs in the Simulation. Larger Numbers Indicate Better Performance

The local case shows that the simulation initially slows down upon the instantiation of cloning, then continues linearly. The uniform case shows a more dramatic slow-down but then proceeds linearly as well. The local cloned case performs significantly faster than the uniform case, once the performance stabilizes. The uniform case progresses about 25 seconds faster than the replicated case, and the local case is about 75 seconds faster. This is a significant difference for an interactive simulation, where alternatives need to be compared and instantiated dynamically. As one would expect, the plots indicate that cloning is more advantageous for simulations using local interactions.

To illustrate how cloning spreads to other logical processes a second metric is evaluated, namely the ratio of messages that are not shared between physical processes with the same version number. The ratio r is given by V_1/V_0, where V_1 and V_0 is the number of events computed by the newly cloned LP and the original LP respectively. A result of $r = 0$ implies the cloned LP have performed no event computations (all computations have been shared between cloned LP and its parent) and $r = 1$ implies that the cloned LP and the original LP have performed the same number of computations. A nonzero value r indicates that the LP has been cloned.

The plot of V_1/V_0 show how cloning spreads to other logical processes over a simulation period. Figure 8 shows the event ratio of non-shared computations for P-Hold in the uniform case and Figure 9 shows it for the local case. Each line in the graph is a snapshot

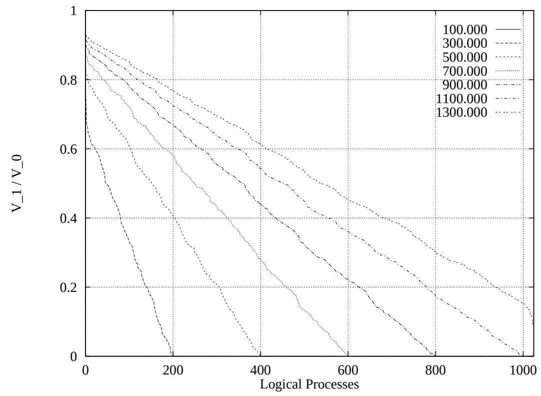

Figure 8: Event Computation Ratio Distribution of P-Hold Dominated by Local Communications. Lower Values Indicate More Shared Computations

at a specific virtual time of the distributed event ratio of all logical processes within the simulation. The uniform P-Hold clones all its logical processes within the first 6 simulated time units after cloning (the spreading occurs between cloning at time 100 and the completion of cloning of all processes at simulated time 106). The local distribution takes more than 1100 simulated time units to clone all its logical processes.

The change in linear progress observed in Figures 7 and 6 reflects the spreading of the cloning. The faster the spreading the quicker the slow-down. In the local case the slope is constant after cloning has spread to close to three quarters of the logical processes, or at

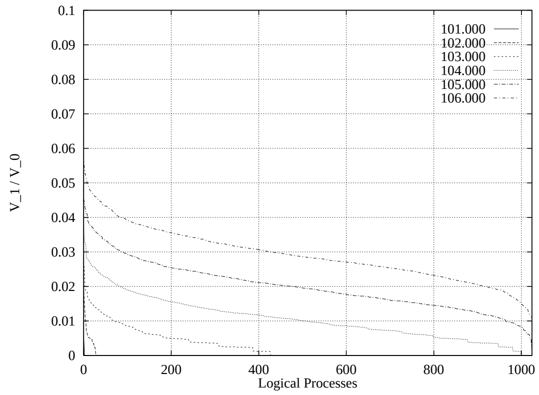

Figure 9: Event Computation Ratio Distribution of P-Hold With Uniformly Distributed Communications. Lower Values Indicate More Shared Computations

900 simulated time units. Similarly, the slope in the uniform case is constant 6 time units after the instantiation of the initial clone which is also when over three quarters of the processes have been cloned. Figure 10 shows the plot of a simulation that magnifies the region between simulated time 90 and 110 of Figure 7, here the slow-down after the initial cloning at time 100 can be observed. After cloning has completed the slopes of both the replicated and the incremental scheme is half of the single clone simulation after all the logical processes have been cloned in both the local and uniform cases. This is not surprising since in the former there are two simulations as opposed to one in the latter.

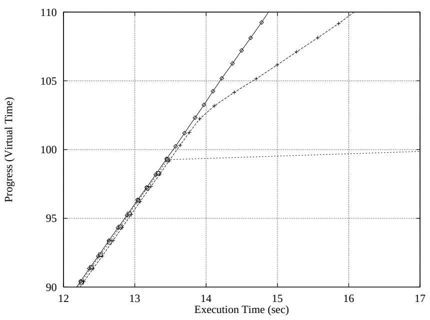

Figure 10: Local P-Hold Performance in the Region between Simulated Time 90 and 110

The performance of cloning using the PCS benchmark is shown in Figure 11. PCS behaves similar to local P-Hold initially, but then degrades toward the replicated PCS simulation. At wall clock time 150 second the replicated simulation surpasses the cloned simulation in performance. One factor impacting performance is a change in the rollback rate. Figure 13 shows rate of roll-backs as the simulation progresses. The graph shows that PCS's rollback rate suddenly drops

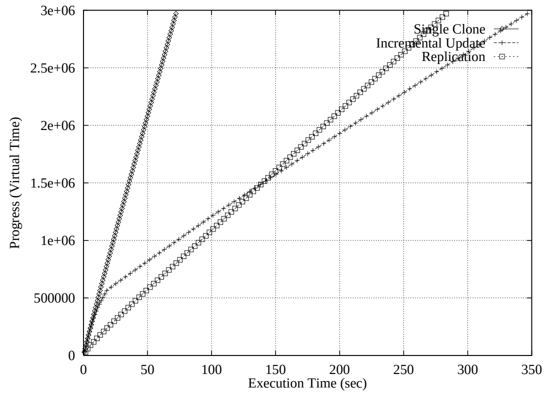

Figure 11: Performance of PCS. Larger Numbers Indicate Better Performance

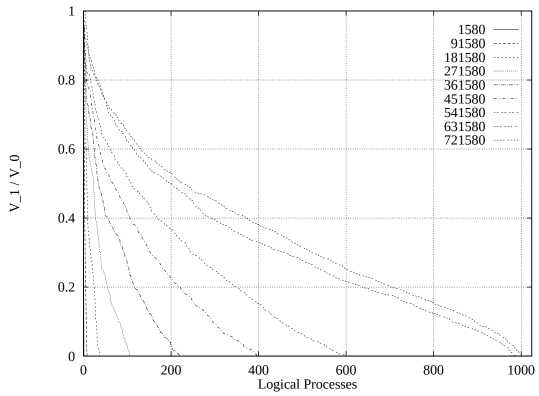

Figure 12: Distribution of Event Computation Ratio of PCS. Lower Values Indicate More Shared Computations

after the initiation of the cloning event then rapidly increases until it reaches simulated time $700,000$ to stabilize. PCS may become unstable because the cloned LPs starts inducing additional rollbacks in un-cloned LPs. This change is dramatic because the rollback rate is typically low, and a small change makes a significant impact. For example, in both of the single cloned cases of P-Hold the rollback rate is over 1000 rollbacks per second, but in the PCS the rollback rate is below 150 rollbacks per second. This behavior, and development of countermeasures are currently under investigation.

These observations suggest that cloning with incremental update may be advantageous for applications that have a low spreading factor and a moderate to high rollback rate. It may be beneficial for application that require the analysis of many alternatives.

6 Conclusion and Future Work

This paper evaluated an incremental cloning mechanism that allows for the exploration of different possible futures in interactive parallel simulation environ-

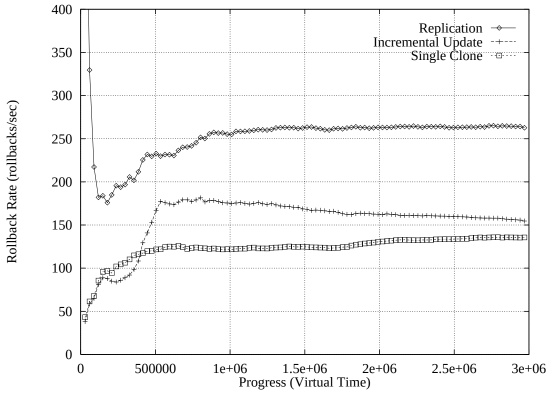

Figure 13: Rollback Rate of PCS

ments. The mechanism is applicable to conservative and optimistic simulation protocols. To avoid copying the *entire* state upon instantiating a clone, the state is cloned incrementally. This is done by assigning the state of a virtual logical process to a physical logical process.

Performance results suggest that the scheme is 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. The cloning scheme may also be beneficial for applications that need to explore many different alternatives, because they can take advantage of the cumulative time-saving of each clone.

Our goal is to demonstrate the effectiveness of the model by implementing cloning with an air traffic control delay application in order to predict airline behavior. For this application it is envisioned that cloning is extended with a mechanism that supports live-feeds of real-time data. A method for re-merging cloned physical LPs if they later converge is being explored. We are also planning to define metrics that characterize simulations which are amenable to cloning.

Currently, it is assumed that a user can interject decision points into the simulator. Another area of future research is defining a mechanism for automating the interjection of decision points, perhaps, by balancing the "running ahead" and "branching" depending on the probability of particular branches against available computational resources and real time constraints. This may offer an effective tool to explore the ordering of simultaneous events [13] and in the analysis by stochastic methods.

Acknowledgments

This work was supported by U.S. Army Contract DASG60-95-C-0103 funded by the Ballistic Missile Defense Organization and the MITRE Corporation. We are thankful to the anonymous referees for their helpful comments.

References

[1] Bestavros, A. 1994. Multi-version speculative concurrency control with delayed commit. In *Proceedings of the 1994 International Conference on Computers and their Applications*, Long Beach, California.

[2] Carothers, C., R. Fujimoto, Y-B. Lin, and P. England. 1994. In *Proceedings of the 1994 MASCOTS Conference.*

[3] Fujimoto, R. M. 1990. In *Proceedings of the SCS Multiconference on Distributed Simulation* 22: 23-28.

[4] Franks, S., F. Gomes, B. Unger, and J. Cleary. 1997. State saving for interactive optimistic simulation. In *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*, 72-79. Burg Lockenhaus, Austria.

[5] Goldberg, A. P. 1992. Virtual time synchronization of replicated processes. In *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, 107-16. Newport Beach, California,

[6] Glasserman, P., P. Heidelberger, and P. Shahabuddin. 1996. Splitting for rare event simulation: Analysis of simple cases. In *Proceedings of the 1996 Winter Simulation Conference*, 302-308. Coronado, California.

[7] Hybinette, M., and R. Fujimoto. 1997. Cloning: a Novel Method for Interactive Parallel Simulation. In *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, Georgia.

[8] Jefferson, J. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems* 7 (3): 404-425.

[9] Vaikili, P. 1992. Massively parallel and distributed simulation of a class of discrete event systems: A different perspective. *ACM Transactions on Modeling and Computer Simulations* 2 (3): 214-238.

[10] Schneider, F. 1990. Implementing fault-tolerance services using the state machine approach: A tutorial. *ACM Computer Surveys* 22 (4): 299-320.

[11] Steinman, J. 1991. SPEEDES: Synchronous parallel environment for emulation and discrete event simulation. In *Proceedings of the SCS Western Simulation Multi-Conference on Advances in Parallel and Distributed Simulation*, 95-103. Anaheim, California.

[12] von Neumann, J. 1956. Probabilistic logics and the synthesis of reliable organism from unreliable components. In *Autonomous Studies*, edited by C. E. Shannon and J. McCarthy. 43-98. Princeton University Press.

[13] Wieland, F. 1997. The threshold of event simultaneity. In *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*, 56-59. Burg Lockenhaus, Austria.

[14] Zayas, E. 1987. Attacking the Process Migration Bottleneck. In *Proceedings of the 11th ACM Symposium on Operating System Principles*, 13-24.