# A Cost Model for Selecting Checkpoint Positions in Time Warp Parallel Simulation

Francesco Quaglia

**Abstract**—Recent papers have shown that the performance of Time Warp simulators can be improved by appropriately selecting the positions of checkpoints, instead of taking them on a periodic basis. In this paper, we present a checkpointing technique in which the selection of the positions of checkpoints is based on a checkpointing-recovery cost model. Given the current state $S$, the model determines the convenience of recording $S$ as a checkpoint before the next event is executed. This is done by taking into account the position of the last taken checkpoint, the granularity (i.e., the execution time) of intermediate events, and using an estimate of the probability that $S$ will have to be restored due to rollback in the future of the execution. A synthetic benchmark in different configurations is used for evaluating and comparing this approach to classical periodic techniques. As a testing environment we used a cluster of PCs connected through a Myrinet switch coupled with a fast communication layer specifically designed to exploit the potential of this type of switch. The obtained results point out that our solution allows faster execution and, in some cases, exhibits the additional advantage that less memory is required for recording state vectors. This possibly contributes to further performance improvements when memory is a critical resource for the specific application. A performance study for the case of a cellular phone system simulation is finally reported to demonstrate the effectiveness of this solution for a real world application.

**Index Terms**—Parallel discrete-event simulation, checkpointing, rollback-recovery, time warp, optimistic synchronization, performance optimization, cost models.

## 1 Introduction

In parallel discrete-event simulation, distinct parts of the system to be simulated are modeled by distinct logical processes (LPs) having their own local simulation clocks [6]. An LP executes a sequence of events and each event execution possibly schedules new events to be executed at a later simulated time. LPs schedule events among each other by exchanging messages carrying the content and the occurrence time (timestamp) of the event. In order to ensure correct simulation results, synchronization mechanisms are used to maintain a nondecreasing timestamp order for the execution of events at each LP; this is also referred to as *causality*. These mechanisms are, in general, conservative or optimistic. The conservative ones enforce causality by requiring LPs to block until certain safety criteria are met. Instead, in the optimistic mechanisms, events may be executed in violation of timestamp order as no "block until safe" policy is considered. Whenever a causality error is detected, a recovery procedure is invoked. This allows the exploitation of parallelism anytime it is possible for causality errors to occur but they do not.

We focus on the Time Warp optimistic mechanism [10]. It allows each LP to execute events unless its pending event set is empty and uses a checkpoint-based rollback to recover from timestamp order violations. A rollback recovers the state of the LP to its value immediately prior the violation. While rolling back, the LP undoes the effects of the events scheduled during the rolled back portion of the simulation. This is done by sending an antimessage for each event that must be undone. Upon the receipt of an antimessage that cancels an event already executed, the recipient LP rolls back as well.

There exist two main checkpointing methods to support state recovery, namely incremental state saving and sparse checkpointing.[1] The former [2], [24], [26] maintains a history of before-images of the state variables modified during event execution so that state recovery can be accomplished by retraversing the logged history and copying before-images into their original state locations. This solution has the advantage of low checkpointing overhead whenever small fractions of the state are updated by event execution. However, in order to provide short state recovery latency, it requires the rollback distance to be sufficiently small. The second method, namely sparse checkpointing, is traditionally defined as recording the LP state periodically, each $\chi$ event execution [12]. If a value of $\chi$ greater than one is used, then the checkpointing overhead is kept low, however, an additional time penalty is added to state recovery. More precisely, state recovery to an unrecorded state involves reloading the latest checkpoint preceding that state and reupdating state variables through the replay of intermediate events (coasting forward). It has been shown [4], [20] that periodic checkpoints taken each $\chi$ event execution give rise to coasting forward with length uniformly distributed between 0 and $\chi - 1$ events.

Recent papers [17], [18] have shown that it is possible to achieve fast state recovery with less checkpointing overhead than that of periodic checkpointing if an appro-

• *The author is with the Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza," Via Salaria 113. 00198 Roma, Italy. E-mail: quaglia@dis.uniroma1.it.*

*Manuscript received 2 Apr. 1999; revised 24 May 2000; accepted 1 Dec. 2000. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 109513.*

1. Recently, solutions mixing features of both methods have been presented in [5], [9], [16], [23].

priate selection of checkpoint positions is adopted. This also has the potential to reduce memory usage of Time Warp, thus possibly improving performance when memory is a critical resource. Along this line, we present in this paper a checkpointing technique in which the selection of the positions of checkpoints relies on a cost model that associates with the current state of the LP a checkpointing-recovery overhead. The checkpointing overhead is either the time to take a checkpoint or zero, depending on whether that state is recorded or not. The recovery overhead is the time penalty associated with a possible future rollback to that state. This penalty varies depending on whether the state is recorded or not. If the state is not recorded, then it must be reconstructed through coasting forward, so the recovery overhead depends on the position of the last taken checkpoint and on the granularity (execution time) of coasting forward events. Then, the convenience of recording the current state as a checkpoint before the execution of the next event is determined using the cost model.

In order to solve the model, we need an estimate of the probability for the current state to be eventually restored due to rollback and we need to know the granularity of the intermediate events from the last taken checkpoint. To estimate the probability value, we present a solution requiring negligible computational effort. Then, we discuss possible solutions to keep track of the granularity of the intermediate events at low cost. This points out the practical viability of our checkpointing technique.

We report an empirical evaluation organized in two parts. In the first, we use a common synthetic benchmark, namely PHOLD [7], to compare our solution with classical (adaptive) periodic techniques. In this part, a number of benchmark parameters are varied, such as the message population, the message routing, the size of state vectors, and the type of the granularity of simulation events (deterministic vs. stochastic). The results of the first part outline the potential of our technique in reducing the completion time of the simulation and, in some cases, in reducing the amount of memory used. Then, in the second part, we report performance data for the case of a cellular phone system simulation. We selected this real world problem as target application for sparse checkpointing because of two main reasons: 1) Typically, there is high communication locality among the LPs hosted by the same processor, therefore, the rollback pattern shows infrequent, long rollbacks that could make the incremental method inadequate; 2) state vectors could have nonminimal size, thus possibly making state saving before the execution of any new event inefficient. The experiments for both parts of the empirical evaluation have been carried out on a cluster of PCs connected through a Myrinet switch. To take full advantage of the communication power of the Myrinet switch we have developed a high speed layer which provides the minimum services needed for a messaging layer to keep high performance.

The remainder of the paper is organized as follows: In Section 2, a background on sparse checkpointing is presented. From Section 3 to Section 5, the cost model and the proposed checkpointing technique are described. The results of the empirical evaluation are reported in Section 6.

## 2 Related Work

The traditional approach to sparse checkpointing consists of recording the LP state periodically, each $\chi$ event execution, $\chi$ being the checkpoint interval. Several analytical models have been presented to determine the time-optimal value ($\chi_{opt}$) for the checkpoint interval. The assumption underlying all these models is that the coasting forward length is uniformly distributed between 0 and $\chi - 1$ events. Simulation results reported in [4], [20] have shown that this is a good approximation of the real distribution of the coasting forward length any time checkpoints are taken on a periodic basis. In addition, most of these models [12], [14], [20] assume there exists a fixed value for the time to record a state as a checkpoint (which is usually a good approximation) and that the granularity of simulation events has small variance. A more general model is the one presented in [22], which takes into account how the exact granularity of simulation events affects the coasting forward time and, thus, the state recovery time. The relevance of this model is in that several real world simulations, such as battlefield simulations or simulations of personal communication systems, actually have high variance of the event granularity which should be taken into account in order to determine the time-optimal value $\chi_{opt}$ of the checkpoint interval.

The extended experimental study in [15] pointed out the effects of the variation of the checkpoint interval on the rollback behavior. Several stochastic queuing networks connected with different topologies were considered. Presented results showed that, when the checkpoint interval slightly increases, a *throttling* effect may appear which tends to reduce the number of rollbacks. This is due to interactions among the LPs on the same processor. Instead, when the checkpoint interval is largely increased, which produces much longer average coasting forwards, a *thrashing* effect may give rise to an increase in the number of rollbacks. This is due to interactions among LPs on distinct processors.

Several adaptive techniques that dynamically adjust the value of $\chi$ have been proposed in order to cope with dynamic rollback behavior which can be originated by a number of causes. Among these, there are possible variations of the load on the processors and possible phase behavior of the LPs in the lifetime of the simulation. Most of these techniques [1], [4], [20] are based on the observation of some parameters of the LP behavior (for example, the rollback frequency) over a certain number of executed events, referred to as *observation period*, and recalculate the checkpoint interval at the beginning of each period. A different approach can be found in [13], where the

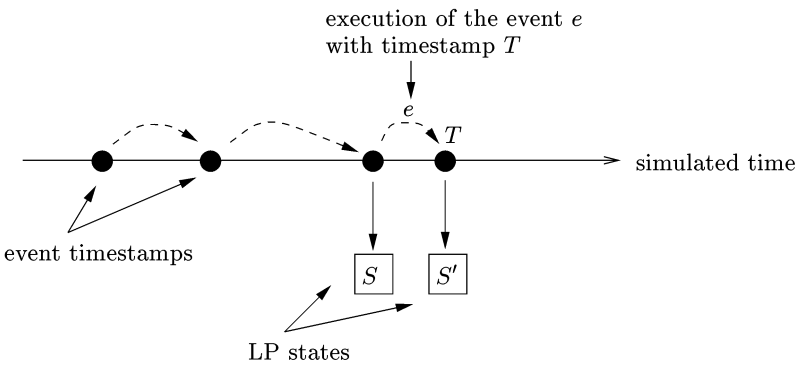


Fig. 1. The LP moves from $S$ to $S'$ due to the execution of $e$.

recalculation of $\chi$ is executed every *Global-Virtual-Time* (GVT) evaluation.[2]

Recently, two papers [17], [18] have shown that it is possible to reduce the overhead due to checkpointing and state recovery by carefully selecting the positions of checkpoints, instead of taking them periodically. The method in [17] takes the checkpoint decision on the basis of the observation of differences between timestamps of two consecutive events. Whenever the execution of an event is going to determine a *large* simulated time increment, then a checkpoint is taken prior to the execution of that event. This solution implicitly assumes that if the LP moves from the state $S$ to the state $S'$, then the probability that $S$ will be eventually restored due to rollback is proportional to the simulated time increment while moving to $S'$. Although this assumption is suited for several simulations [3], [17] it has never been extensively tested; this limits the generality of the solution. The method in [18] is based on a notion of *probabilistic* checkpointing which works as follows: For any state $S$, an estimate of the probability that it will have to be restored due to rollback is performed, namely $P_e(S)$. Then, before moving from $S$ to $S'$, a value $\alpha$ uniformly distributed in the interval [0, 1] is extracted and a checkpoint of $S$ is taken if $\alpha > 1 - P_e(S)$. In other words, $S$ is recorded with probability equal to $P_e(S)$, therefore, the higher the probability that $S$ will have to be restored, the higher the probability that it is recorded as a checkpoint. What we noted in this method is that: 1) The probabilistic decision is actually *memoryless* as it does not take into account the position of the last taken checkpoint to establish if $S$ must be recorded or not and, in addition, 2) the decision itself is independent of the real cost of saving the state vector of the LP. Specifically, if a checkpoint has been taken a few events ago, then $S$ can be reconstructed through coasting forward without incurring a significant time penalty (this is true especially in the case of small grain coasting forward events), therefore, it would be convenient to not record $S$ as a checkpoint even if the probability $P_e(S)$ is not minimal (this is true especially in the case of large size of the state vector). This is not captured by the probabilistic approach.

The checkpointing technique we propose in this paper solves the latter problem as the cost model it relies on takes into account the position of the last taken checkpoint and, also, the real cost of saving the state vector of the LP. In addition, as already pointed out in the introduction, the recovery overhead associated with any state is computed by explicitly considering the granularity of any event involved in a (possible) coasting forward and not just a mean granularity value. The latter feature allows our solution to exhibit the potential for providing good performance in case of both small and large variance of the event granularity.

## 3 Selecting Checkpoint Positions

In this section, we present the cost model and the associated policy for selecting the positions of checkpoints. Then, we discuss operational issues related to the tracking of the granularity of the intermediate events from the last taken checkpoint and to the estimation of the probability for the current state to be eventually restored. We recall these two quantities are required to solve the cost model.

### 3.1 The Cost Model and the Selection Policy

The LP moves from one state to another due to the execution of simulation events. An example of this is shown in Fig. 1, where the arrow extending toward the right-end represents simulated time, black circles represent event timestamps, and labeled boxes represent state values at given points in the simulated time, that is, those points corresponding to event timestamps. In our example, the LP moves from the state $S$ to $S'$ due to the execution of the event $e$ with timestamp $T$. We associate with each state $S$ passed through by the LP a probability value, namely $P(S)$, which is the probability that $S$ will have to be restored due to a future rollback. $P(S)$ will be used in the construction of the cost model expressing the checkpointing-recovery overhead associated with $S$.

Denoting with $\delta_s$ the time to save or reload the LP state vector, which is assumed to be constant, as in most previous analyses (see [12], [14], [20], [22], [23]), the checkpointing

2. The GVT is defined as the lowest value among the timestamps of events either not yet executed or currently being executed or carried on messages still in transit. The GVT value represents the commitment horizon of the simulation because no rollback to a simulated time preceding GVT can ever occur. The GVT notion is used to reclaim memory allocated for obsolete messages and state information and to allow operations that cannot be undone (e.g., I/Os, displaying of intermediate simulation results, etc.). The memory reclaiming procedure is known as *fossil collection*.

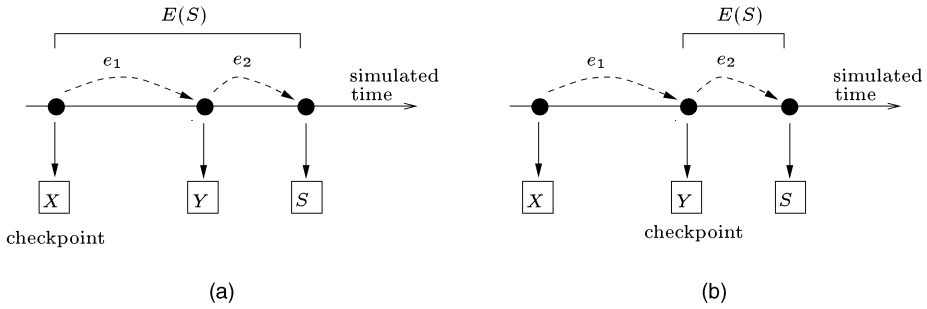


Fig. 2. Two examples for $E(S)$.

overhead $C(S)$ associated with the state $S$ can be expressed as follows:[3]

$$C(S) = \begin{cases} \delta_s & \text{if } S \text{ is recorded} \\ 0 & \text{if } S \text{ is not recorded.} \end{cases} \quad (1)$$

Expression (1) points out that if $S$ is recorded as a checkpoint, then there is a checkpointing overhead associated with it which is quantified by the time to take a checkpoint $\delta_s$.

Let us now model the recovery overhead. Before proceeding in the discussion, we remark that this overhead expresses only the latency to recover to the state $S$ as a function of the checkpointing activity of the LP; it does not take into account the effects of sending antimessages in the rollback phase. We denote as $E(S)$ the set of all the events that move the LP from the latest checkpointed state preceding $S$ to $S$. For the example shown in Fig. 2a, the latest checkpointed state preceding $S$ is $X$ and $E(S) = \{e_1, e_2\}$. Instead, for the example in Fig. 2b, the latest checkpointed state preceding $S$ is $Y$ and $E(S) = \{e_2\}$. Denoting with $\delta_e$ the granularity (execution time) of the event $e \in E(S)$, then we can associate with the state $S$ the following recovery overhead $R(S)$:[4]

$$R(S) = \begin{cases} P(S)\delta_s & \text{if } S \text{ is recorded} \\ P(S)[\delta_s + \sum_{e \in E(S)} \delta_e] & \text{if } S \text{ is not recorded.} \end{cases} \quad (2)$$

Expression (2) states that, in the case of rollback to $S$ (this happens with probability $P(S)$), if $S$ is recorded as a checkpoint, then the recovery overhead consists only of the time $\delta_s$ to reload $S$ into the current state buffer. Otherwise, it consists of the time $\delta_s$ to reload the latest checkpoint preceding $S$, plus the time to replay all the events in $E(S)$, that is, the coasting forward time.

We denote as $CR(S)$ the checkpointing-recovery overhead associated with $S$. It results as the sum of $C(S)$ and $R(S)$. Therefore, combining (1) and (2), we get for $CR(S)$ the following expression:

$$CR(S) = \begin{cases} \delta_s + P(S)\delta_s & \text{if } S \text{ is recorded} \\ P(S)[\delta_s + \sum_{e \in E(S)} \delta_e] & \text{if } S \text{ is not recorded.} \end{cases} \quad (3)$$

Expression (3) represents our cost model. Using this model, we introduce below a selection policy for determining the positions of checkpoints. Basically, the selection policy is such that the state $S$ is recorded as a checkpoint before the execution of the next event if such recording results in the minimization of the value of $CR(S)$. More technically, denoting with $CR(S)_y$ the value of $CR(S)$ in case $S$ is recorded and, with $CR(S)_n$, the value of $CR(S)$ in case $S$ is not recorded, then the selection policy can be synthesized by the following expression:

**Selection-Policy (SP)**
before moving from $S$:
**if** $CR(S)_y \leq CR(S)_n$
**then** record $S$
**else** do not record $S$

Note that, while defining $R(S)$ and, thus, $CR(S)$, we have implicitly assumed that the probability $P(S)$ does not change depending on whether $S$ is recorded or not. More technically, it is assumed that the checkpointing actions do not affect the rollback behavior. This assumption has some resemblance to what is assumed in some periodic checkpointing techniques [20], [22], [23], although there are some differences. Specifically, in these techniques, it is assumed that small changes in the checkpoint interval $\chi$ will (possibly) result in small changes in the rollback behavior. Instead, in the construction of our cost model, we have assumed that taking or not a checkpoint in a specific point will result in no perceptible change in the rollback behavior. However, we argue that taking or not a checkpoint in a specific point will ultimately result in shorter or longer average distance between checkpoints, which has resemblances to choosing shorter or longer values for the checkpoint interval in classical periodic techniques.

With regard to responsiveness of the checkpointing technique to real variations of the rollback behavior, classical adaptive, periodic solutions have the property that they detect if rollbacks become more frequent and decrease the checkpoint interval $\chi$ in order to avoid rollback thrashing. This can be done promptly by selecting an adequately short observation period, after which the value of the checkpoint interval is adjusted. For the case of **SP**, we have a similar behavior. Specifically, if the value of $P(S)$ becomes larger, then both $CR(S)_n$ and $CR(S)_y$ grow.

3. In the present analysis, we use the same value $\delta_s$ for both the state saving time and the time to reload a recorded state vector into the current state buffer as this is a common, realistic assumption. However, the analysis can be easily extended to the cases in which this assumption is not verified.

4. Recall that $\delta_e$ takes into account only the time to execute the event $e$; it does not take into account the time to send out new events possibly scheduled during the execution of $e$. Therefore, $\delta_e$ expresses exactly the time to replay $e$ in coasting forward as no event is sent out in such phase.

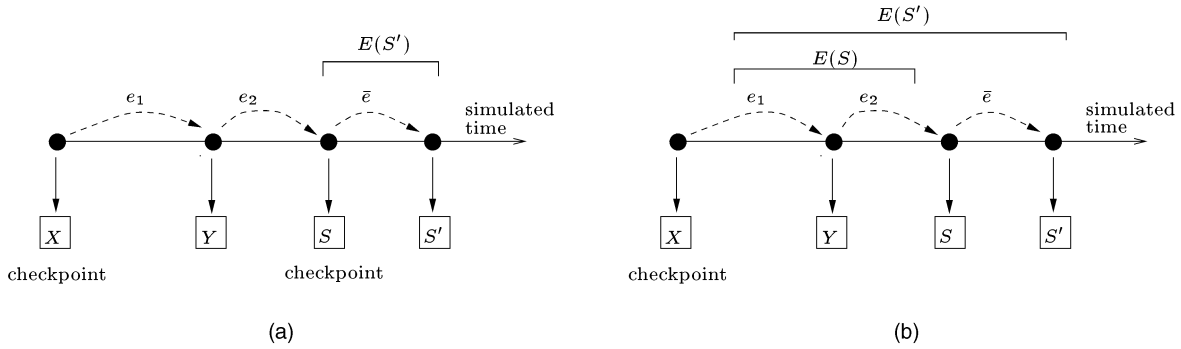


Fig. 3. Effects of the Recording of $S$ on $CR(S')$.

However, computing the first differential of $CR(S)$ vs. $P(S)$, it can be seen that the growth of $CR(S)_n$ is larger than that of $CR(S)_y$, therefore, for larger values of $P(S)$, **SP** is likely to reduce the average distance between checkpoints, thus avoiding rollback thrashing.

Concerning operational issues, solving the cost model (i.e., computing $CR(S)_y$ and $CR(S)_n$) requires knowledge of:

1. the state saving/reloading time $\delta_s$;
2. the granularity of the intermediate events from the last taken checkpoint, namely the events belonging to $E(S)$; and;
3. the probability $P(S)$ for the current state $S$ to be eventually restored.

The parameter $\delta_s$ is typically known upon the execution of the simulation as it depends on the size (number of bytes) of the state vector and on the time per byte needed for recording it on the hardware/software architecture used. Instead, both the granularity of the intermediate events and the probability $P(S)$ typically depend on proper dynamics of the specific application. In what follows, we discuss possible solutions for tracking on-line the granularity of the intermediate events (Section 3.2) and we present a solution to estimate the value of $P(S)$ (Section 3.3). Before presenting these solutions, let us discuss two peculiarities of **SP**.

#### 3.1.1 Maximal Knowledge Exploitation

From among several parameters, **SP** determines the convenience of recording $S$ as a checkpoint prior to the execution of the next event using information on:

1. the position of the last taken checkpoint (which determines the events that are in $E(S)$);
2. the exact granularity of the events executed from the last taken checkpoint (those events in $E(S)$).

We remark that information in points 1) and 2) actually encodes the *maximal knowledge* related to the portion of the simulation already executed and to past checkpointing actions, which is relevant to establishing the amount of recovery overhead associated with $S$ in case $S$ is not recorded and a rollback to it occurs. More precisely, the positions of the checkpoints other than the latest one preceding $S$ and the granularity of any event out of the set $E(S)$ do not affect the time to reconstruct $S$ in case it is not recorded.

#### 3.1.2 Local vs. Global Overhead Minimization

Let us consider the portion of the simulation execution shown in Fig. 3. $S'$ immediately follows $S$ and the event which moves the LP from $S$ to $S'$ is $\bar{e}$. In case $S$ is recorded as a checkpoint (see Fig. 3a), the set $E(S')$ contains only the event $\bar{e}$, so the checkpointing-recovery overhead $CR(S')$ associated with the state $S'$ is:

$$CR(S') = \begin{cases} \delta_s + P(S')\delta_s & \text{if } S' \text{ is recorded} \\ P(S')[\delta_s + \delta_{\bar{e}}] & \text{if } S' \text{ is not recorded.} \end{cases} \quad (4)$$

Instead, if $S$ is not recorded as a checkpoint (see Fig. 3.b), then the set $E(S')$ contains the same events as the set $E(S)$ ($e_1$ and $e_2$ in our example) plus the event $\bar{e}$. Therefore, the checkpointing-recovery overhead $CR(S')$ becomes as follows:

$$CR(S') = \begin{cases} \delta_s + P(S')\delta_s & \text{if } S' \text{ is recorded} \\ P(S')[\delta_s + \sum_{e \in E(S)} \delta_e + \delta_{\bar{e}}] & \text{if } S' \text{ is not recorded} \end{cases} \quad (5)$$

(note that to also derive expressions (4) and (5), we suppose $P(S')$ independent of checkpointing actions).

The previous example points out that the outcoming decision of **SP** on whether the state $S$ must be recorded or not determines the shape of the function $CR$ associated with the states that will follow $S$. This implies that taking the checkpoint decision based only on the minimization of the checkpointing-recovery overhead associated with the current state of the LP could not lead to the minimization of the whole checkpointing-recovery overhead of the simulation, that is, the one resulting from the sum of the checkpointing-recovery overheads associated with all the states passed through in the course of the simulation. However, we recall that such (global) minimization requires the knowledge of unknown information, such as the exact sequence of states that will be passed through. Overall, **SP** selects the *best* checkpoint positions with respect to the portion of the simulation already executed.

### 3.2 Tracking the Granularity of the Intermediate Events

A key point concerning operational issues associated with the policy **SP** is the tracking of the granularity of the intermediate events from the last taken checkpoint. Before entering a discussion on this point, we recall that parallel discrete-event technology is typically adopted for models

with event granularity ranging from several tenths or hundreds of microseconds up to some milliseconds or more on current conventional architectures.[5] Therefore, the tracking mechanism should provide precision in the order of $1\mu s$.

At first glance one might think the tracking mechanism could be implemented by using system calls that report values of the CPU utilization time. Unfortunately, this approach is likely to not fit requirements of parallel simulation applications. More precisely, in conventional operating systems technology, the accounting mechanism for the CPU utilization time is implemented using a software counter updated by timer interrupts with period $10ms$, which is the most used scheduling time slice. As a consequence, system calls accessing data structures recording accounting information provide precision on the order of $10^4$ $\mu s$, which results are inadequate.

The previous problem can be solved in different ways depending on whether we can assume: 1) The parallel simulation application is the only active application on the computing system (i.e., the computing system is dedicated to this application) and 2) message passing is totally implemented at the application level (i.e., no kernel or demon process activity is involved in message passing operations).[6]

If assumptions in points 1) and 2) hold, then the granularity of a given event can be approximated with minimal error by means of the real time elapsed between the start and the end of the event execution. Errors can be caused by timer interrupts (occurring each $10ms$) that fall within the execution interval of the event. However, we note that timer interrupts are typically handled within a few microseconds, therefore, for the case of large grain events (on the order of hundreds of microseconds up to some milliseconds), the error due to the interrupt handler is negligible in practice. In addition, for the case of small grain events (on the order of several tenths of microseconds), the likelihood of timer interrupt falling within the execution interval of the event is expected to be minimal. Overall, if we measure the event granularity by means of elapsed real time, then the interrupt handler is expected to cause very little overestimation of the granularity of a minimal percentage of the events.

Real time measures can be obtained by using light system calls allowing access to hardware real time clocks (implemented as hardware counters). For both Unix and Linux systems, there exists a system call, namely `gettimeofday()`, that returns real time measures with precision on the order of $1\mu s$. Using two calls to `gettimeofday()`, one at the beginning of the event execution, the other at the end, we can evaluate the elapsed real time comprised between the start and the end of the event execution. Furthermore, the overhead due to these system calls is negligible in practice. As support to this argument, we report in Table 1 the time consumption for the execution of `gettimeofday()` for the case of a Pentium II 300 MHz (this is the type of machine we have used for the empirical study in Section 6) and a Pentium III 450 MHz, both running Linux, a Sparc Station Ultra 2, and a Sparc Station Ultra 10 both running Solaris. To obtain the values reported (expressed in microseconds), we have run a simple benchmark, structured as a loop, where the call is executed $10^8$ times (in this way, we can get measures on the order of tenths or hundredths of seconds, which practically eliminate errors due to fluctuations). The values in the table are computed as the ratio between the time for the loop, and $10^8$. Hence, these values represent upper bound values to the real time consumption for a single call to `gettimeofday()` since they account also for the instructions to manage the control variable of the loop.

TABLE 1
Time per Call to `gettimeofday()`

| Pentium II | Pentium III | Ultra 2 | Ultra 10 |
|---|---|---|---|
| 2.1 | 1.0 | 0.4 | 0.3 |

By these results we get that the overhead due to the tracking mechanism is bounded by about $4\mu s$ for the Pentium II, $2\mu s$ for the Pentium III, and by less than $1\mu s$ for the other architectures, thus pointing out how, for event granularity values of at least several tenths or hundredths of microseconds, the overhead actually results negligible.

If assumptions in points 1) and 2) do not hold, the tracking mechanism based on real time high resolution clocks is not feasible since the elapsed real time between two successive calls to `gettimeofday()` could be an incorrect measure of the granularity of the event. In other words, the correctness of the measure depends on the scheduling sequence of kernel, demon, and application processes activities. A solution to cope with this problem consists of implementing an accounting mechanism at the application level by embedding it into the event code. This can be realized by adding, at the end of each block consisting of a sequence of instructions that are either all executed or all skipped, a single accounting instruction incrementing a software counter that keeps track of the microseconds for the execution of that block on the specific hardware/software architecture. As an example, in Fig. 4a, we have an instruction sequence with a conditional statement at a given point. The instruction sequence can be seen as the composition of three distinct blocks, see Fig. 4b. Two of these blocks are associated with the branches of the conditional statement. For this code structure, we can add three accounting instructions, as shown in Fig. 4c, in the way that each accounting instruction is associated with a specific block. At the end of the block execution, the accounting instruction is activated. Obviously, the accounting instructions associated with the branches of the conditional statement should also take into account the cost to evaluate the conditional expression. Similar approaches can be adopted for the case of sequences of instructions within a loop and so on. The overhead due to this embedded tracking mechanism

5. For models with lower event granularity, that is, a few tenths of microseconds or less, it would be preferable to run multiple copies of the simulation sequentially on the available machines as a different, more convenient, form of parallelization.

6. As we will describe in Section 6.1, the testing environment we have used to evaluate **SP** is such that message passing operations are totally controlled at the application level, thus requiring no kernel or demon process activity.

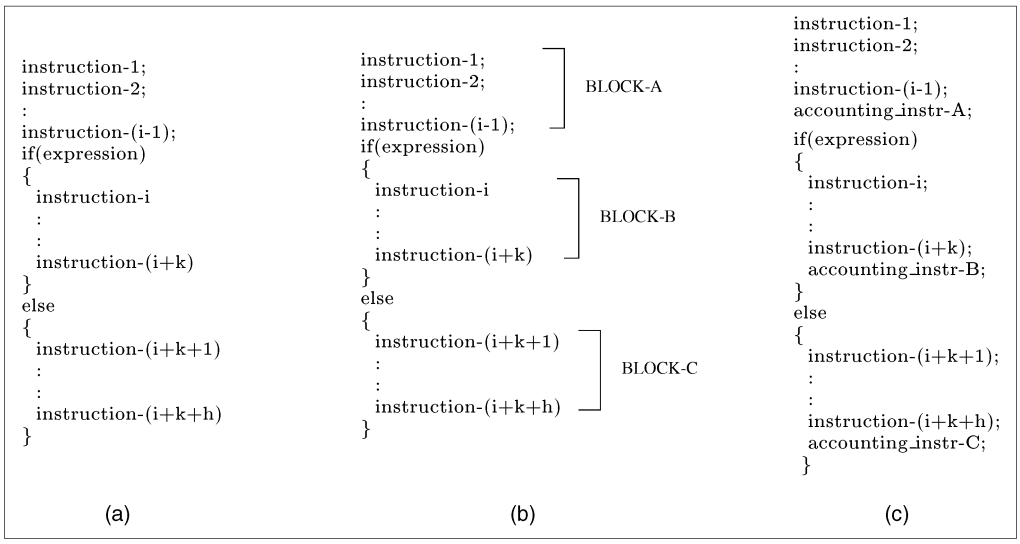


Fig. 4. An example for the insertion of accounting instructions.

depends on the amount of accounting instructions for a given execution path, which, in turn, depends on the particular structure of the simulation code. Nevertheless, unless the code structure contains a very small number of instructions per block, it is expected that the overhead due to the accounting mechanism is negligible in practice. In addition, we note that any accounting instruction involves only the updating of a software counter that can be implemented as an integer; this operation is not time consuming. This file could be generated automatically prior to compiling the simulation code by running benchmarks for measuring the time consumption for the instruction blocks on the specific hardware/software architecture.

For what concerns transparency to the user, the responsibility to implement the tracking mechanism pertains to the software designer. This is the same requirement of other classical solutions. As an example, the adaptive, periodic technique presented in [20] needs knowledge of the average execution time for the events of any LP. This depends on the specific type of the events occurring at that LP and also on how possible conditional branches or loops in the code structure actually affect the average event execution time for the specific application. In general, this is unknown before the simulation execution as the way conditional branches or loops are executed depends on proper dynamics of the application. In addition, there also exists the possibility that the average execution time for the events occurring at an LP follows a phase behavior. Tracking this behavior should be done automatically during the execution in a transparent way to the user, therefore, a form of on-line tracking mechanism should be embedded in the code structure by the designer.

### 3.3 Estimating Probability Values

As pointed out in Section 3.1, the solution of the cost model (i.e., computing the values of $CR(S)_y$ and $CR(S)_n$) needs knowledge of the probability $P(S)$. In this section, we present a method to estimate this value. The method has resemblances to those presented in [3], [18].

Let $sc(S)$ denote the value of the simulation clock associated with the state $S$ and let $e$, with timestamp $ts(e)$, be the event which moves the LP from $S$ to its subsequent state. The execution of the event $e$ produces an increment in the simulation clock of the LP, moving it from $sc(S)$ to $ts(e)$. Then, with any state $S$, we can associate a simulated time interval, namely $I(S)$, whose length is $ts(e) - sc(S)$, which is delimited as follows:

$$I(S) = (sc(S), ts(e)]. \tag{6}$$

The probability $P(S)$ corresponds to the probability that a rollback will occur in the simulated time interval $I(S)$. Recall that a rollback in the interval $I(S)$ occurs either because events are scheduled later with timestamps in that interval (i.e., after $e$ is executed, an event $e'$ such that $sc(S) < ts(e') < ts(e)$ is scheduled for the LP) or because the LP that scheduled the event $e$ rolls back revoking $e$ (i.e., the antimessage for $e$ arrives after $e$ is executed).

We perform an estimate of $P(S)$ based on the length of the interval $I(S)$ and on the monitoring of the relative frequency of rollback occurrence in simulated time intervals of a given length. We define simulated time points $t_i$ such that: 1) $t_i = 0$ if $i = 0$ and 2) $t_i < t_{i+1}$. Then, we decompose the simulated time positive semiaxis into intervals $I_i = [t_i, t_{i+1})$. For each state $S$, there exists an interval $I_i$ such that the length of $I(S)$, namely $L(I(S))$, is within that interval (i.e., $L(I(S)) \in I_i$).

For each interval $I_i$, the LP keeps two counters, namely $N_i$ and $R_i$, initially set to zero. $N_i$ counts the amount of event executions associated with intervals $I(X)$ such that $L(I(X)) \in I_i$.[7] $R_i$ counts the amount of rollback occurrences in simulated time intervals $I(X)$ such that $L(I(X)) \in I_i$. Using these counters, we estimate the probability $P(S)$ as:

7. Coasting forward events are not counted by $N_i$. This is because they are not real simulation events since they are an artifact of the state recovery procedure.

$$\text{if } L(I(S)) \in I_i \text{ then } P(S) = \frac{R_i}{N_i}. \quad (7)$$

Note that if the decomposition has a unique interval $I_0 = [0, \infty)$, then $P(S)$ is computed as the ratio between the total number of rollbacks occurring at the LP and the total number of event executions. This ratio is commonly referred to as the rollback frequency of the LP. Obviously, with this type of decomposition there is the implicit assumption that all the states are equally likely to be restored, therefore, the real rollback pattern of the LP could be not fully exploited to make **SP** effective. How to determine a suitable decomposition will be discussed in the following subsection. Then, we present a discussion on the real impact of the quality of the estimation method on the effectiveness of **SP**.

### 3.3.1 Determining the Decomposition

Choosing $t_i$ points to determine the decomposition of the simulated time positive semiaxis should be done taking into account proper dynamics of the specific application, such as the average simulation clock advancement at the LP. Typically, this parameter depends on the proper nature of the application (e.g., types of events and their density in the simulated time, types of the distributions of the timestamp increment determining the timestamps of new events dynamically produced, etc.), therefore, it does not result as viable in practice to determine a priori a suitable decomposition for any arbitrary simulation. By this argument, the decomposition should be determined at run time and, similarly to what happens for the tracking mechanisms discussed in Section 3.2, the designer of the simulation software should embed in the code structure the instructions to determine the decomposition in a totally transparent way to the user. In this section, we shall discuss two different approaches for the determination of the decomposition:

**Uniform approach.** In this solution, each interval $I_i$ has length equal to $\Delta$ except the last one, namely $I_{last} = [t_{last}, +\infty)$. The value $last$ determines the number of intervals of the decomposition (i.e., the amount of memory destined to the counters $N_i$ and $R_i$). To determine $\Delta$, the LP should initially monitor its simulation clock advancement to track the expected value of the advancement. Then, $\Delta$ could be selected as a fraction (e.g., 1/5, 1/10, or less) of the expected value, and the value of $last$ could be selected in order to get a value for $t_{last} = \Delta \times last$ in the order of the maximum clock advancement observed. The decomposition could be recalculated periodically in order to prevent the last selected values for $\Delta$ and $last$ from becoming inadequate due to relevant variations of the advancement pattern of the simulation clock of the LP in the lifetime of the simulation.

With this type of decomposition, the individuation of the counter to be updated each time an event is executed or a rollback occurs and the individuation of the counters to estimate probability values are simple and introduce negligible overhead. For example, when a rollback occurs to a state $X$, the index $i$ of the counter $R_i$ to be updated is easily computed as follows:

$$i = \begin{cases} \lfloor \frac{L(I(X))}{\Delta} \rfloor & \text{if } L(I(X)) < t_{last} \\ last & \text{if } L(I(X)) \geq t_{last.} \end{cases} \quad (8)$$

As opposed to its simplicity, the main drawback of this approach is in that the data-point density is typically nonuniform. As a consequence, counters associated with distinct intervals could exhibit different statistical significance. This should not reveal a problem if at least a minimum amount of observations are collected for each interval. Anyway, to tackle this issue, nonuniform decompositions could be used as we shall discuss below.

**Non-uniform Approach.** Let us denote as $F$ the distribution function of the simulation clock advancement of the LP, then $t_i$ points originating evenly balanced observations in different intervals can be determined by using the reverse function $F^{-1}$. Specifically, in a decomposition with $n$ intervals, $t_i$ points can be calculated as $t_i = F^{-1}(i/(n+1))$. The big problem associated with this solution is that the function $F$ is typically unknown.

The simplest way to overcome this problem consists of approximating $F$ with an exponential function. This approximation is justifiable in practice since, in general, events originating small simulation clock advancement are more likely compared to events originating large advancement. Therefore, a decomposition based on exponential assumption should be revealed as appropriate in most of the cases. Anyway, even if the exponential assumption is not completely verified in practice, it's more likely to get almost balanced data-points using the exponential assumption than adopting a uniform decomposition. Note, in addition, that the exponential assumption provides the reverse function directly (this property is not verified for other classical distributions). To determine the exact shape of $F$ for the specific application, the expected value of the simulation clock advancement must be estimated. Therefore, similarly to the case of uniform decomposition, the LP should initially perform monitoring actions on the clock advancement. Once the estimate for the expected simulation clock advancement, namely $esca$, is available, the points of the decomposition are computed as

$$t_i = -\frac{log(1 - i/(n+1))}{1/esca}.$$

The decomposition could be recalculated periodically to cope with relevant variations of $esca$.

If no exponential assumption is made, then the estimate of $F$ must be performed on-line. Although this solution is more general, it could introduce unacceptable overhead. Recall in addition that, even if the estimate of $F$ is performed efficiently, computing the reverse function could severely impact the final performance. Therefore, this approach could be revealed as infeasible in practice, especially for the cases in which it could be mandatory to recalculate the decomposition several times during the simulation execution.

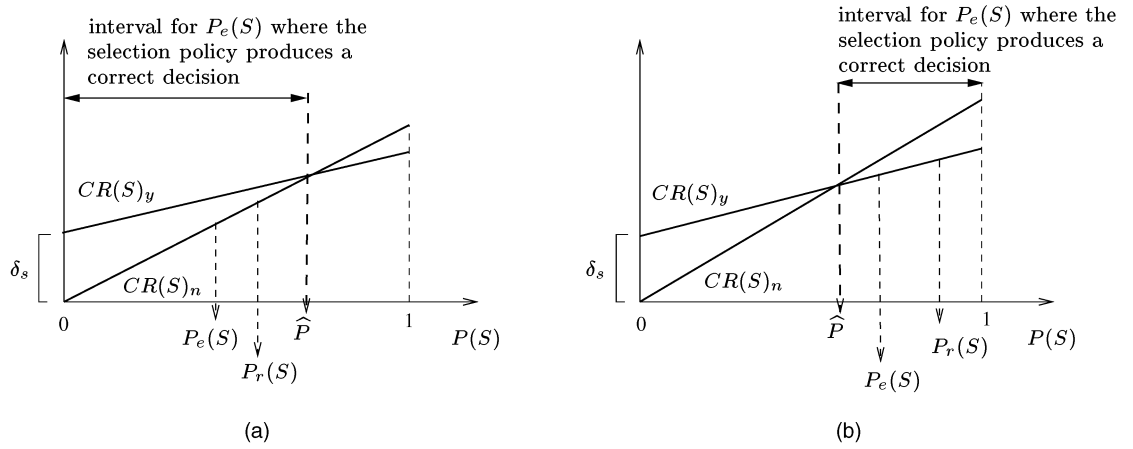


Fig. 5. Gerneral cases for the functions $CR(S)y$ and $CR(S)n$.

Beyond the possibility of recalculating the decomposition in case of relevant variations of the average simulation clock advancement, there also exists the possibility that the rollback pattern changes in time even with no relevant variation of the average clock advancement. One approach to tackle nonstationarity in the rollback pattern exploits the common belief that, in most simulations, the near past behavior is a good approximation of the near future behavior. Therefore, probability values can be estimated using data-points related to a temporal window. A few hundred of events usually constitute a window length producing reliable results [4], [18], [20].

### 3.3.2 A Discussion on the Real Impact of the Quality of the Estimation Method

In this section, we show by discussion that low quality estimates of probability values have no relevant impact on the effectiveness of **SP**. So, the simple estimation method based on counters previously introduced represents an adequate solution.[8]

Before proceeding in the discussion, we introduce the following simple notion: We say that **SP** leads to a *wrong* decision anytime the checkpoint decision based on the estimated value of $P(S)$ is different from the one that would have been obtained by considering the real value of $P(S)$. Otherwise, we say that **SP** produces a *correct* decision.

In Fig. 5a and in Fig. 5b we show two cases for the linear functions $CR(S)_y$ and $CR(S)_n$ vs. the value of $P(S)$. Recall that, when $P(S)$ is equal to zero, $CR(S)_n$ is equal to zero and $CR(S)_y$ is equal to $\delta_s$. The cases shown are general as the two functions have an intersection point within the interval [0, 1) for $P(S)$. A more particular situation is obtained when $CR(S)_n < CR(S)_y$ in the whole interval $0 \leq P(S) < 1$, that is, when $CR(S)_y$ and $CR(S)_n$ do not intersect or intersect for $P(S) = 1$. We denote as $\widehat{P}$ the value of $P(S)$ corresponding to the intersection point; in addition, we denote as $P_r(S)$ the real value of $P(S)$ and with $P_e(S)$ the corresponding estimated value.

Suppose $P_r(S) < \widehat{P}$ (see Fig. 5a), in this case $CR(S)_y > CR(S)_n$, so there is no real convenience of recording $S$ as a checkpoint. The same decision is taken by **SP** for any estimated value $P_e(S)$ less than $\widehat{P}$. Therefore, for any value $P_e(S) < \widehat{P}$, **SP** always produces a correct decision. Suppose $P_r(S) \geq \widehat{P}$ (see Fig. 5b), in this case $CR(S)_y \leq CR(S)_n$, so there is a real convenience of recording $S$. The same decision is taken by **SP** for any estimated value $P_e(S)$ larger than or equal to $\widehat{P}$. Therefore, for any value $P_e(S) \geq \widehat{P}$, **SP** always produces a correct decision. Overall, **SP** produces a correct decision any time one of the following two cases occurs:

C1: Both $P_r(S)$ and $P_e(S)$ are lower than $\widehat{P}$;
C2: Both $P_r(S)$ and $P_e(S)$ are higher than or equal to $\widehat{P}$.

Instead, it produces a wrong decision anytime one of the following two cases occurs:

C3: $P_r(S) < \widehat{P}$ and $P_e(S) \geq \widehat{P}$;
C4: $P_r(S) \geq \widehat{P}$ and $P_e(S) < \widehat{P}$.

Case **C3** or **C4** may occur if:

1. The value of $P_r(S)$ is close to $\widehat{P}$ (in this case, we may get a wrong decision even with a small distance between $P_r(S)$ and $P_e(S)$; anyway, the distance must be such that it moves $P_e(S)$ to the opposite side of $P_r(S)$ with respect to the value $\widehat{P}$);
2. The value or $P_r(S)$ is far from $\widehat{P}$ and a very large distance exists between $P_r(S)$ and $P_e(S)$ (anyway, the distance must be such that it moves $P_e(S)$ to the opposite side of $P_r(S)$ with respect to the value $\widehat{P}$).

From previous considerations, we argue that, in order for **SP** to produce a wrong decision (cases **C3** and **C4**), a set of conditions must be satisfied, namely those conditions producing situations in points 1) or 2). Therefore, the states for which these conditions are actually satisfied will be, in general, a (small) subset of all the states passed through in the course of the simulation. Hence, for the majority of the

8. The approach, based on counters, has the advantage that it can be implemented at very low cost (this is true especially for the case of uniform decomposition), but it shows the drawback that the estimates could be of limited quality. With respect to this point, no control on the trust of the estimate is performed and no run-time decision as a function of the trust is performed. On the other hand, high quality estimates could be produced by using more complex statistical methods that might produce probing effects. Therefore, in general, this is not a feasible solution.

states, **SP** will produce correct decisions.[9] This "robustness" of **SP** derives from the fact that it maps values of a continuous function, that is, the difference between $CR(S)_y$ and $CR(S)_n$, into a Boolean domain. This mapping into such a discrete domain removes the effects of "noise" (i.e., the effects of low quality estimate of probability values) unless the noise itself oversteps a given threshold.

## 4 Checkpointing vs. Fossil Collection Frequency

Sparse checkpointing can interfere with the frequency of GVT calculation and fossil collection whose objective is to recover memory allocated for obsolete state information and messages. Specifically, as rollback to simulated time equal to GVT is possible, then, in order to correctly support state recovery, each LP must retain the latest recorded state (i.e., the latest checkpoint) with simulated time $T$ less than or equal to GVT and also all the messages carrying events with a timestamp larger than $T$. Therefore, that checkpoint and all those messages cannot be discarded during the execution of the fossil collection procedure. If very few states are recorded in the course of the simulation, then it is possible that a large amount of messages must be retained. The drawback incurred is that the amount of memory recovered during any fossil collection may be small, thus there is the risk that the GVT calculation and the fossil collection procedure must be executed frequently (as memory saturates frequently). This may have detrimental effects on performance.

Adopting **SP**, there is the possibility that very few states are recorded in the course of the simulation execution. This may happen whenever, for any state $S$, the value of the probability $P(S)$ approaches the value zero. As an extreme case, if $P(S)$ is exactly equal to zero for any state $S$, then **SP** will never induce the LP to take a checkpoint. This leads **SP** to possibly produce a negative interference with the frequency of GVT calculation and fossil collection. In order to prevent this risk, we allow the LP to take (rare) periodic checkpoints. To this purpose, the LP maintains two integer variables, namely $max\_dist$ and $event\_ex$. The variable $max\_dist$ records the maximum number of event executions allowed between two consecutive checkpoint operations. The variable $event\_ex$ represents the current distance, in terms of events, from the last checkpoint operation. By using these two variables, **SP** is modified as follows:

**Modified-Selection-Policy (MSP)**
before moving from $S$:
**if** $(CR(S)_y \leq CR(S)_n) \vee (event\_ex = max\_dist)$
**then** record $S$
**else** do not record $S$

9. Recall, in addition, that there exists a set of states for which cases **C3** and **C4** can never occur independently of the distance between the real and the estimated probability values. These are all the states $S$ such that $CR(S)_n < CR(S)_y$ in the whole interval [0, 1) for $P(S)$. For any of these states, either the two functions $CR(S)_n$ and $CR(S)_y$ do not intersect or they intersect in the point $\widehat{P} = 1$, therefore, cases **C3** and **C4** cannot occur as neither $P_r(S)$ nor $P_e(S)$ can be higher than one. For all these states, **SP** always produces a correct decision.

**MSP** does not allow the distance between two consecutive checkpoints to be larger than $max\_dist$ events, that is, $max\_dist$ state transitions, thus avoiding the negative interference with GVT calculation and fossil collection frequency. Adaptive, periodic techniques tackle this problem adopting default values for the maximum checkpoint interval $\chi_{max}$ which usually are between 15 and 30 (see [4], [20]). Any value within that interval will be well-suited for $max\_dist$.

## 5 A Final Description of the Checkpointing Technique

There is just another point to be fixed in order to provide a final, complete description of our checkpointing technique. The modified policy **MSP** (just like the original **SP**) relies on the solution of the cost model which needs the estimate of probability values. This means that the policy cannot be applied if at least a few statistical data are not available (note that the problem of the absence of statistical data for the selection of the initial value of the parameter(s) proper to the checkpointing technique is a common problem to almost all existing adaptive techniques [4], [17], [18], [20]). To overcome this problem, we partition the execution of the LP into two main phases, namely A and B. Phase A consists of few hundred events. During this phase, the LP collects statistical data to estimate probability values and records as checkpoints all the states passed through. This can be done by adopting **MSP** with $max\_dist$ initially set to one. During Phase B, the LP continues to collect statistical data (possibly updating dynamically the decomposition and/or using a windowing mechanism for the counters) and takes the checkpoint decision using **MSP** with a value for $max\_dist$ selected within the interval [15, 30].

Since it is typical of most simulations that the very early behavior of the LP at the simulation starting is not a good indicator of the immediate future behavior, it does not result to be convenient to use statistical data related to this initial phase. For this reason, we introduce an additional phase, namely Start-Up, in which **MSP** is adopted with $max\_dist$ set to one, and no statistical data are collected. In Fig. 6, the complete behavior of the LP is reported.

## 6 A Performance Study

In this section, we report a two-part performance study of **MSP**. The first part was conducted using the PHOLD model in several different configurations. The second part was conducted using a cellular phone system simulation model; this part demonstrates the effectiveness of our solution for the case of a real world application. Prior to presenting the results, we describe the testing environment, list the checkpointing techniques we have selected for the comparison with **MSP**, and introduce the performance parameters we have considered.

### 6.1 Testing Environment

The experiments were all performed on a cluster of four PCs Pentium II 300 MHz (128 Mbytes RAM) running LINUX as the operating system, interconnected by a high speed Myrinet switch based on wormhole technology. This

Start-Up (few hundred events)

**settings:** $max\_dist := 1$

**actions:** select checkpoint positions using **MSP**

Phase A (few hundred events)

**settings:** $max\_dist := 1$

**actions:** collect statistical data; select checkpoint positions using **MSP**

Phase B (till the end of the simulation)

**settings:** select $x \in [15, 30]$; $max\_dist := x$

**actions:** collect statistical data; select checkpoint positions using **MSP**

Fig. 6. Behavior of an LP (only the actions relevant to checkpointing are shown).

type of architecture is actually an emerging one for parallel applications due to cost vs. performance reasons and also to expandability/modifiability.

Any PC is connected to the Myrinet switch through an interface implemented on a card consisting of a LANai processor equipped with local memory and supports for DMA. The LANai's memory is mapped into the address space of the host PC, therefore, it can be accessed directly or using DMA. The LANai processor runs a *control program* that performs send and receive operations. This program can be designed according to requirements of the specific application. Depending on the structure of the control program and of the associated message passing layer run at the host PC, messages at the receive side can be buffered into host PC memory or into the memory on-board of the interface card and then transferred on demand into the host memory. We have developed a high speed layer, namely Minimal Fast Messages (MFM), tailored for optimizing the delivery delay of small size messages. These layer results are well-suited for parallel simulation applications where the amount of data associated with message/antimessage transmission is typically small. In MFM, the buffering at the receive side is done into the on-board memory of the interface card, therefore, messages/antimessages are transfered into the host memory only when a receive operation is issued by the application. This operation performs the copying of the message from the memory on-board of the interface card to a buffer in the address space of the application process. On the other hand, a send operation issued by the application simply involves the copy of the message content onto the on-board memory of the interface card. Then, the LANai processor will transfer the message toward the destination. Therefore, using MFM, message passing operations at any host are totally controlled at the application level (with no kernel or demon process activity involved). This feature, combined with the fact that, during the experiments, the machines were completely dedicated to the parallel simulation application, allowed us to implement the tracking mechanism for the granularity of the intermediate events from the last taken checkpoint using calls for the access to hardware real time clocks.

In our Time Warp system, the events are implemented as a compound structure with several fields (sender, receiver, timestamp, etc.). For the case of PHOLD, the structure has total size 36 bytes; for the case of the mobile phone system, it has total size 44 bytes. Using MFM, any message carrying an event is delivered within about 30 $\mu s$ for both PHOLD and the cellular phone system. The same delays characterize the transmission of antimessages. Message exchange among LPs hosted by the same machine does not involve operations of the MFM layer. There is an instance of the Time Warp kernel on each processor. The kernel manages the local event list (resulting as the logical collection of the input queues of the local LPs) and schedules LPs for event execution according to the Smallest-Timestamp-First algorithm [11]. Memory space for new entries into the input and output queues of the LPs is allocated dynamically using classical `malloc()` calls. Therefore, there is no pool of preallocated buffers. Instead, preallocated buffers have been used for entries of the stack of saved state vectors. The cancellation phase is implemented following the aggressive policy, that is, antimessages are sent as soon as the LP rolls back [8]. Fossil collection is executed periodically. Rollback is nonpreemptive.

### 6.2 Selected Checkpointing Techniques

In order to evaluate the benefits from **MSP** we have selected as reference solutions two checkpointing techniques:

- Classical Periodic State Saving (PSS) with fixed value for the checkpoint interval $\chi$;
- The adaptive, periodic solution by Ronngren and Ayani (RA) [20]. As already discussed before, RA is based on an analytical model that defines the value of the time-optimal checkpoint interval (the assumptions underlying the model have been recalled in Section 2). The model is used to recalculate the value of the checkpoint interval $\chi$ based on the observed variations of the rollback frequency. Specifically, at the end of each observation period, the new value for $\chi$ is calculated as a weighted sum of the old value and the time optimal value defined by the model as a function of the rollback frequency observed in the last period. An upper limit on $\chi$ is introduced so as

not to interfere with the frequency of fossil collection. In our experiments, we used the parameter setting defined by the authors for the weights of the two terms of the sum, the observation period length and also the upper limit on $\chi$.

We did not consider any incremental state saving method in our comparison. This is because previous studies ([14], [19], [21]) have already pointed out that incremental state saving and sparse checkpointing outperform each other in distinct classes of simulation problems, so the two approaches are effective in distinct domains.[10]

In the comparison, we also consider the case of Copy State Saving (CSS), namely a checkpoint before the execution of each simulation event. Results for CSS are relevant to point out the real gain achievable through sparse checkpointing techniques, therefore, simulations with CSS act as control simulations.

For what concerns PSS, we report the results for the case of the observed time-optimal value of $\chi$. For the case of **MSP**, we have adopted a uniform decomposition with $\Delta$ fixed at 1/5 of the initially monitored average clock advancement of the LP. The decomposition is calculated once (no periodic recalculation is performed). A windowing mechanism is used in order to compute the estimate using statistical data that refer to the last 1,000 executed events at most. After the initial transient behavior, the value of $max\_dist$ is set to 30.

### 6.3 Performance Parameters

We report measures related to the following parameters:

- The *average checkpoint interval*, that is, the average number of executed events between two successive checkpoint operations. This parameter is representative of the average checkpointing overhead per event;
- The *rollback frequency* and the *average rollback distance*. These parameters, combined together, point out whether the performance difference between different checkpointing techniques derives from the proper nature of the techniques or from changes in the rollback pattern. Specifically, if two techniques show the same final values for rollback frequency and average rollback distance, then none of them outperforms the other due to variations in the rollback pattern. Therefore, possible performance gains of one technique over the other actually derive from the proper nature of the techniques. The values of the rollback frequency and of the average rollback distance allow us to also test the real feasibility of assuming no significant effect of the checkpointing technique on the rollback pattern;
- The *event rate*, that is, the number of committed events per time unit. This parameter indicates how fast the simulation execution is with a given checkpointing technique, therefore, it is representative of the final performance perceived.

In addition, we report some data to point out the maximum memory utilization, expressed in Mbytes, under different checkpointing techniques.[11] Note that, in the very early part of the simulation, both RA and **MSP** behave like CSS. Therefore, the maximum memory utilization of both RA and **MSP** would be similar to that of CSS if statistical data related to this part would be included. In order to point out the real reduction of the maximum memory utilization of RA and **MSP** as compared to CSS at steady state, statistical data related to the very early part are not included. Finally, we report the speedup over the sequential execution. This parameter is an indicator of the efficiency of the parallel implementation.[12]

For each configuration of the benchmark, we report the average observed values of previous parameters, computed over 10 runs that were all done with different seeds for the random number generation. For the case of the event rate, we also report the standard deviation computed over the 10 samples in order to demonstrate the statistical significance of the final performance results. At least $5 \times 10^6$ committed events have been simulated in each run.

### 6.4 Results for the PHOLD Model

The PHOLD model, originally presented in [7], consists of a fixed number of LPs and of a constant number of jobs (messages) circulating among the LPs (that is referred to as job population). Both the routing of jobs among the LPs and the timestamp increments are taken from some stochastic distributions. Although a set of standard benchmarks for parallel discrete-event simulation does not exist, PHOLD is, in practice, one of the most used ones.

For this benchmark, we have considered a basic configuration, namely symmetric PHOLD, and three configurations derived from the basic one by varying several parameters such as the event granularity (deterministic vs. stochastic) and the message routing (static vs. dynamic). Furthermore, for each configuration, we have considered two different job populations and two distinct sizes for the state vectors. The detailed descriptions of the configurations and the associated results are separately reported in each of the following subsections.

#### 6.4.1 Symmetric PHOLD

In this configuration, the PHOLD model is composed of 64 homogeneous LPs evenly distributed among the four machines of the cluster. The timestamp increment is exponentially distributed with mean 10 simulated time units for all the LPs and jobs are equally likely to be forwarded to any other LP. The two job populations selected are: 1) one job per LP and 2) 10 jobs per LP. The granularity of any event is deterministic and is fixed at 140 $\mu s$. Due to this deterministic nature, no tracking

10. The incremental method is preferable for simulations with very large state size, very small portions of the state updated by event execution and short rollback distances. For any other simulation setting, sparse checkpointing provides better performance.

11. We recall that the average memory utilization cannot be observed without interfering with the simulation execution. Instead, the maximum memory utilization (i.e., the maximum amount of memory destined for keeping checkpoints and messages) can be easily measured.

12. The sequential simulator we have used adopts preallocated buffers for the entries of the event list. Adopting preallocated buffers also in the Time Warp simulator should be likely to yield better speedup results as compared to those reported in this study.

TABLE 2
Results for Symmetric PHOLD

**1 Job per LP - 2 Kbytes State Vector Size**

| Checkpointing Technique | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.151 | 1.29 | 10587 (88) | 25.3 | 1.55 |
| PSS | 3 | 0.153 | 1.28 | 12735 (97) | 9.2 | 1.87 |
| RA | 3.52 | 0.156 | 1.25 | 12525 (109) | 8.9 | 1.84 |
| MSP | 3.80 | 0.154 | 1.28 | 14020 (137) | 9.0 | 2.06 |

**1 Job per LP - 4 Kbytes State Vector Size**

| Checkpointing Technique | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.158 | 1.32 | 8774 (93) | 41.6 | 1.29 |
| PSS | 4 | 0.160 | 1.30 | 12110 (114) | 14.8 | 1.78 |
| RA | 4.57 | 0.162 | 1.31 | 11718 (92) | 12.8 | 1.72 |
| MSP | 4.98 | 0.161 | 1.30 | 13291 (118) | 13.0 | 1.95 |

**10 Jobs per LP - 2 Kbytes State Vector Size**

| Checkpointing Technique | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.052 | 1.97 | 13607 (113) | 32.5 | 2.00 |
| PSS | 5 | 0.049 | 2.01 | 15629 (137) | 7.9 | 2.30 |
| RA | 5.59 | 0.049 | 2.02 | 15843 (178) | 7.5 | 2.33 |
| MSP | 6.12 | 0.050 | 2.00 | 16654 (142) | 6.8 | 2.51 |

**10 Jobs per LP - 4 Kbytes State Vector Size**

| Checkpointing Technique | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.049 | 1.96 | 10566 (102) | 50.1 | 1.56 |
| PSS | 7 | 0.048 | 1.98 | 14586 (152) | 10.3 | 2.14 |
| RA | 7.89 | 0.048 | 2.01 | 15555 (138) | 9.9 | 2.29 |
| MSP | 10.66 | 0.049 | 1.97 | 16586 (161) | 7.9 | 2.44 |

mechanism is activated at all. Overall, this configuration allows us to test the checkpointing techniques when considering no variance for the event granularity and also exact a priori knowledge of the event granularity. The two distinct sizes for the state vectors are: 1) 2 Kbytes and 2) 4 Kbytes. Two Kbytes state vectors are saved in approximately $\delta_s = 70\ \mu s$; 4 Kbytes state vectors are saved in approximately $\delta_s = 140\ \mu s$. The case of 2 Kbytes state size models simulations with medium/small state granularity (with respect to the event granularity), whereas the case of 4 Kbytes state size models simulations with medium/large state granularity. The results are reported in Table 2.

#### 6.4.2 Symmetric PHOLD with Distinct Job Types

This configuration has the same features as the symmetric PHOLD described in Section 6.4.1 except for what concerns the event granularity. There are three distinct types of jobs, namely $a$, $b$, and $c$. The three types have deterministic granularity $\delta_a = 50\ \mu s$, $\delta_b = 150\ \mu s$, and $\delta_c = 220\ \mu s$, respectively. The type of job is selected from among the three job types according to a uniform distribution. After a job is served (but before it is forwarded to another LP), the job type is redefined by uniformly selecting it in the set $\{a, b, c\}$. Therefore, with probability 1/3, the type of the job remains unchanged when it is forwarded to another LP. As for previous configuration, the deterministic granularity value for each event type does not require the activation of the on-line tracking mechanism. This configuration, allows us to test the checkpointing techniques when considering nonminimal difference in the granularity of different event types, with perfect a priori knowledge of the granularity for each type of event. For this configuration, we considered the same two distinct job populations and the same two distinct state vector sizes as for the symmetric PHOLD in Section 6.4.1. The results are reported in Table 3.

#### 6.4.3 Symmetric PHOLD with Stochastic Event Granularity

This configuration has the same features as the symmetric PHOLD in Section 6.4.1 except for what concerns the event granularity. Event granularity is not deterministic but stochastic. Specifically, the granularity of any event is exponentially distributed with mean $140\ \mu s$. The nondeterministic nature of the event granularity needs the activation of the on-line tracking mechanism. This configuration allows us to test the checkpointing techniques when considering nonminimal variance for the event granularity with no exact a priori knowledge of the event granularity. The same two distinct job populations and the same two distinct state vector sizes of previous configurations were considered. The results are reported in Table 4.

#### 6.4.4 Asymmetric PHOLD with Dynamic Routing

This configuration has the same features as the symmetric PHOLD in Section 6.4.1 except for what concerns the routing of jobs among the LPs. There are four hot spot LPs to which 30 percent of all jobs must be routed. The hot spot LPs change in the course of the simulation (they change

TABLE 3
Results for Symmetric PHOLD with Distinct Job Types

**1 Job per LP - 2 Kbytes State Vector Size**

| **Checkpointing Technique** | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.155 | 1.29 | 10502 (92) | 25.1 | 1.60 |
| PSS | 3 | 0.158 | 1.28 | 12035 (138) | 10.0 | 1.88 |
| RA | 3.48 | 0.159 | 1.27 | 11787 (108) | 8.8 | 1.79 |
| MSP | 3.77 | 0.157 | 1.28 | 13470 (94) | 9.0 | 2.05 |

**1 Job per LP - 4 Kbytes State Vector Size**

| **Checkpointing Technique** | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.165 | 1.32 | 8381 (71) | 40.2 | 1.27 |
| PSS | 4 | 0.170 | 1.30 | 11663 (101) | 14.3 | 1.77 |
| RA | 4.70 | 0.172 | 1.29 | 11393 (84) | 12.1 | 1.73 |
| MSP | 4.98 | 0.170 | 1.30 | 13112 (108) | 12.3 | 1.96 |

**10 Jobs per LP - 2 Kbytes State Vector Size**

| **Checkpointing Technique** | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.050 | 1.95 | 13351 (112) | 32.1 | 2.10 |
| PSS | 5 | 0.049 | 2.00 | 15425 (131) | 7.9 | 2.42 |
| RA | 5.41 | 0.048 | 2.01 | 15652 (136) | 7.6 | 2.46 |
| MSP | 5.96 | 0.049 | 2.01 | 16502 (147) | 6.8 | 2.64 |

**10 Jobs per LP - 4 Kbytes State Vector Size**

| **Checkpointing Technique** | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.050 | 1.97 | 10536 (103) | 50.2 | 1.65 |
| PSS | 7 | 0.050 | 1.99 | 14776 (142) | 10.8 | 2.31 |
| RA | 7.78 | 0.049 | 2.01 | 15407 (105) | 9.8 | 2.41 |
| MSP | 10.22 | 0.051 | 1.93 | 16452 (131) | 8.5 | 2.58 |

TABLE 4
Results for Symmetric PHOLD with Stochastic Event Granularity

**1 Job per LP - 2 Kbytes State Vector Size**

| **Checkpointing Technique** | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.157 | 1.29 | 10483 (106) | 26.0 | 1.58 |
| PSS | 3 | 0.161 | 1.28 | 12562 (135) | 10.4 | 1.89 |
| RA | 3.34 | 0.164 | 1.27 | 12447 (130) | 9.3 | 1.88 |
| MSP | 3.59 | 0.165 | 1.27 | 13904 (158) | 9.5 | 2.1 |

**1 Job per LP - 4 Kbytes State Vector Size**

| **Checkpointing Technique** | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.159 | 1.33 | 8704 (90) | 41.6 | 1.32 |
| PSS | 4 | 0.161 | 1.32 | 12012 (87) | 14.9 | 1.81 |
| RA | 4.40 | 0.164 | 1.29 | 11657 (103) | 13.4 | 1.76 |
| MSP | 4.66 | 0.162 | 1.30 | 13183 (108) | 13.8 | 1.99 |

**10 Jobs per LP - 2 Kbytes State Vector Size**

| **Checkpointing Technique** | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.054 | 2.00 | 13512 (131) | 32.3 | 2.05 |
| PSS | 5 | 0.051 | 1.96 | 15534 (134) | 10.4 | 2.36 |
| RA | 5.35 | 0.052 | 2.01 | 15721 (144) | 8.2 | 2.37 |
| MSP | 5.89 | 0.051 | 2.00 | 16664 (171) | 7.1 | 2.57 |

**10 Jobs per LP - 4 Kbytes State Vector Size**

| **Checkpointing Technique** | Average Checkpoint (Interval) | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.050 | 1.95 | 10442 (97) | 50.6 | 1.59 |
| PSS | 7 | 0.050 | 1.94 | 14473 (109) | 10.7 | 2.20 |
| RA | 7.40 | 0.051 | 1.92 | 15423 (123) | 10.6 | 2.35 |
| MSP | 10.86 | 0.050 | 1.96 | 16466 (168) | 8.1 | 2.50 |

TABLE 5
Results for Asymmetric PHOLD

**1 Job per LP - 2 Kbytes State Vector Size**

| Checkpointing Technique | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.124 | 1.31 | 11208 (92) | 28.3 | 1.65 |
| PSS | 3 | 0.125 | 1.30 | 13079 (127) | 10.9 | 1.92 |
| RA | 3.65 | 0.126 | 1.28 | 13612 (150) | 9.2 | 2.03 |
| MSP | 3.84 | 0.127 | 1.27 | 14687 (137) | 9.4 | 2.16 |

**1 Job per LP - 4 Kbytes State Vector Size**

| Checkpointing Technique | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.125 | 1.30 | 9368 (84) | 42.9 | 1.38 |
| PSS | 4 | 0.127 | 1.29 | 12386 (120) | 15.2 | 1.82 |
| RA | 4.63 | 0.126 | 1.28 | 12934 (122) | 13.6 | 1.90 |
| MSP | 4.97 | 0.125 | 1.31 | 13985 (117) | 13.7 | 2.06 |

**10 Jobs per LP - 2 Kbytes State Vector Size**

| Checkpointing Technique | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.043 | 2.03 | 13970 (142) | 33.6 | 2.08 |
| PSS | 6 | 0.042 | 2.05 | 16175 (118) | 6.9 | 2.41 |
| RA | 6.45 | 0.041 | 2.06 | 16591 (175) | 6.7 | 2.47 |
| MSP | 7.23 | 0.042 | 2.04 | 17601 (149) | 6.0 | 2.62 |

**10 Jobs per LP - 4 Kbytes State Vector Size**

| Checkpointing Technique | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.042 | 2.02 | 10892 (98) | 54.9 | 1.62 |
| PSS | 7 | 0.042 | 2.01 | 14887 (137) | 10.6 | 2.21 |
| RA | 8.23 | 0.043 | 1.99 | 16062 (117) | 9.8 | 2.39 |
| MSP | 11.14 | 0.044 | 1.98 | 17321 (155) | 7.9 | 2.55 |

each $3 \times 10^4$ simulated time units and the sequence of the changes is defined prior to the simulation execution by randomly picking up new hot spots among all the LPs). This configuration possibly gives rise to simulations which do not reach steady state of the rollback behavior. It allows us to test the checkpointing techniques when considering nonuniform, variable message routing among the LPs. The same two job populations and the same two state vector sizes of previous configurations were considered. The results are shown in Table 5.

### 6.4.5 General Comments

The performance data collected for the PHOLD model indicate two major tendencies, regardless of the type of the event granularity, the state vector size, and the message routing. These two tendencies are determined by the two different job populations used.

For the case of low job population (one job per LP), the final performance of **MSP** is between 8 percent and 13 percent better than that obtained using the (adaptive) periodic techniques and the maximum amount of memory used by **MSP** is about the same as that used by RA (the maximum difference is 3-4 percent). CSS and PSS exhibit higher maximum memory utilization. Therefore, the tendency is a relevant performance gain of **MSP** with no relevant difference for what concerns the maximum amount of memory used as compared to RA. Instead, for the case of high job population (10 jobs per LP), the performance gain of **MSP** is lower (between 3 percent and 8 percent) and the maximum amount of memory used by **MSP** is definitely lower than that of the periodic techniques (up to 25 percent as compared to RA and up to 30 percent as compared to PSS). Therefore, the tendency is a sensible performance gain with a relevant reduction of the maximum amount of memory used.

The reason for these different tendencies is in the different frequency of rollback for the two cases. When the job population is low, the rollback frequency is nonminimal (about 12-16 percent), therefore, a large number of state recovery procedures are executed. In this case, the gain of **MSP** derives from two sources: 1) It shows slightly larger average checkpoint interval, which tends to reduce the checkpointing overhead; 2) it is expected to produce shorter state recovery latency due to the wise selection of the positions of checkpoints. As respect to point 2), **MSP** is such that it tends to save an LP state if it has very high probability to be eventually restored, the last checkpoint was taken several events ago, and the intermediate events have large granularity. This tends to reduce the state recovery time as compared to periodic techniques, which, in the case of frequent rollbacks, has a relevant positive impact on the final performance perceived. When the job population grows, the frequency of rollback tends to decrease, therefore, fewer state recovery procedures are executed. In this case, the performance gain of **MSP** is reduced as it derives almost exclusively from the larger average checkpoint interval. In other words, the lower amount of recovery procedures makes the impact of possible reductions in the recovery time on the final performance perceived lower. Nevertheless, the increase

TABLE 6
Results for the Cellular Phone System

| Checkpointing Technique | Average Checkpoint Interval | Rollback Frequency | Average Rollback Distance | Event Rate - (Standard Deviation) | Maximum Memory Utilization | Speedup |
|---|---|---|---|---|---|---|
| CSS | 1 | 0.004 | 38.98 | 10023 (83) | 12.2 | 2.15 |
| PSS | 10 | 0.004 | 41.11 | 11612 (102) | 2.2 | 2.49 |
| RA | 11.27 | 0.004 | 40.34 | 11805 (89) | 2.0 | 2.54 |
| MSP | 16.64 | 0.004 | 39.81 | 12585 (112) | 1.4 | 2.71 |

of the average checkpoint interval of **MSP** is larger as compared to the case of low job population, therefore, a relevant reduction of the memory used is noted (especially when considering medium/large state vector size). Overall, **MSP** shows potential for significant reductions of the completion time of the simulation in case of frequent rollbacks. On the other hand, it shows potential for sensible reductions of the completion time of the simulation and for strong reductions of the amount of memory used in case of infrequent rollbacks.

We note that the rollback pattern, namely frequency and average distance of rollback, is very similar for all the checkpointing techniques. Therefore, the performance difference derives essentially from the proper nature of the techniques. Finally, the speedup achieved with optimized sparse checkpointing is on the order of 50 percent of the ideal one for the case of low job population and on the order of 65 percent of the ideal one for the case of high job population. Therefore, the checkpointing techniques have been tested over an efficient implementation (recall that speedup on the order of 50 percent of the ideal one is considered as a classical threshold determining acceptable performance for the parallel execution).

### 6.5 Results for the Cellular Phone System

Sparse checkpointing techniques look good as solutions for the case of parameterized synthetic benchmarks. However, for many real world applications, they could be revealed as useless. As an example, for the case of queuing network simulations, the distance of rollback is typically very short and either the size of the state vectors is very small or, in case of larger size (this happens for the case of models in which statistics associated with any job moving around the network are maintained and incrementally updated), the fraction of the state updated by the execution of an event is typically small. Therefore, copy state saving or incremental methods represent adequate solutions. As another example, for the case of Time Warp simulations of systems with memoryless components (e.g., combinatorial logic gates), recent work on the concept of rollback relaxation [25] has shown how rollback can be implemented with no underlying state saving mechanism, thus rendering useless in practice any optimized sparse checkpointing technique.

To our knowledge, examples of real world applications where sparse checkpointing can actually be revealed as effective are simulations of wireless systems such as cellular phone systems, paging networks, and personal communication systems (PCSs). For these applications, the rollback pattern typically consists of infrequent and long (up to several tenths of events) rollbacks due to high communication locality among the LPs hosted by the same machine. Furthermore, state vectors could have nonminimal size. For these applications, both copy state saving and incremental methods could provide poor performance. In this section, we report results evaluating the effectiveness of **MSP** for the case of a cellular phone system simulator.

Our simulation model consists of a ring highway with 16 base stations. Each base station is modeled by a single LP and provides a wireless coverage area to mobile phones called *cell*. Each cell has 20 channels allocated to it and covers a portion of the highway of 3,000 meters in length. Call requests arrive to each cell according to an exponential distribution with interarrival time 16 $s$ (therefore, the interarrival time for the whole system is 1 $s$) and the average holding time for each call is two minutes. When a call is issued, the position of the associated mobile phone in the cell is randomly selected according to an uniform distribution. Mobility of the cellular phones within the highway is modeled as a random speed following a truncated Gaussian distribution with mean 80 km/hour. The direction for the movement of the phone is selected uniformly among the two possible directions. All the calls initiated within a given cell are originated by the LP associated with that cell, therefore, no external call generator is used. There are three main types of events, namely hand-off due to mobile phone cell switch, call termination, and call arrival. A call termination simply involves the release of the associated channel and statistics update; the granularity of this type of event is about $40\ \mu s$. The granularity of the events associated with call arrival depends on whether the channels at the destination cell are all busy or not. If all channels are busy, the incoming call is simply counted as a block; if at least one channel is available, then signal to noise ratio must be computed when allocating the channel. Therefore, the granularity of this event type may vary from about $40\ \mu s$ up to $0.6\ ms$. When a hand-off occurs between adjacent cells, the hand-off event at the cell left by the mobile simply involves the release of the channel and has granularity of about $40\ \mu s$. Instead, the granularity of the hand-off event at the destination cell varies, depending on whether all channels in this cell are busy or not. If there is no available channel, then the call is simply cut off (dropped), otherwise, an available channel is assigned to the call and signal-to-noise ratio is computed when allocating the channel. Overall, the granularity for this event type has range similar to that associated with incoming call events. The size of the state vector is little more than 1 Kbyte and the state saving cost is about $40\ \mu s$. Each of the four machines hosts the same number of LPs; the obvious mapping of LPs to machines is adopted. The results, reported in Table 6, are aligned with the data obtained for the PHOLD model. In particular, rollbacks are infrequent, therefore, we expect to observe a sensible performance gain of **MSP** with a significant reduction of the maximum amount of memory used due to larger

average checkpoint intervals. The data confirm this expectation. Finally, we note that, with optimized sparse checkpointing, speedup on the order of 67 percent of the ideal one is achieved.

## 7 Summary

In this paper, we have presented a general solution for tackling the checkpoint problem in Time Warp simulations. The checkpointing technique we have proposed selects the positions of the checkpoints using a cost model which expresses the checkpointing-recovery overhead associated with any state passed through in the course of the simulation. The cost model determines the convenience of recording the current state before the execution of the next event. We have discussed implementation issues related to this technique in order to show its practical viability. In particular, we have presented solutions to evaluate on-line at low cost the quantities involved in the cost model and we have discussed the effectiveness of these solutions.

Simulation results are reported to evaluate and compare this technique with classical (adaptive) periodic techniques. To this purpose, a classical benchmark in several different configurations as well as a real world application have been used. The results show that the proposed technique has the potential to reduce the running time of the parallel simulation application and, in some cases, also the memory usage.

## Acknowledgments

The author thanks Bruno Ciciani for many interesting discussions on the checkpoint problem in Time Warp simulators. A special thanks goes to Alessandro Fabbri for suggestions on the organization of the cellular phone system simulation code and to Luca Becchetti who has allowed the author to run benchmarks on his machine. The author is indebted to the anonymous referees whose comments were crucial to improving the technical soundness and the presentation of the paper.

## References

[1] L.R.G. Auriche, F. Quaglia, and B. Ciciani, "Run-Time Selection of the Checkpoint Interval in Time Warp Based Simulations," *Simulation Practice and Theory,* vol. 6, no. 5, pp. 461-478, July 1998.

[2] H. Bauer and C. Sporrer, "Reducing Rollback Overhead in Time Warp Based Distributed Simulation with Optimized Incremental State Saving," *Proc. 26th Ann. Simulation Symp.,* pp. 12-20, Mar. 1993.

[3] A. Ferscha and J. Luthi, "Estimating Rollback Overhead for Optimism Control in Time Warp," *Proc. 28th Ann. Simulation Symp.,* pp. 2-12, Apr. 1995.

[4] J. Fleischmann and P.A. Wilsey, "Comparative Analysis of Periodic State Saving Techniques in Time Warp Simulators," *Proc. Ninth Workshop Parallel and Distributed Systems (PADS '95),* pp. 50-58, June 1995.

[5] S. Franks, F. Gomes, B.W. Unger, and J. Cleary, "State Saving for Interactive Optimistic Simulation," *Proc. 11th Workshop Parallel and Distributed Systems (PADS '97),* pp. 72-79, June 1997.

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

[7] R.M. Fujimoto, "Performance of Time Warp under Synthetic Workloads," *Proc. Multiconference Distributed Simulation,* vol. 22, no. 1, Jan. 1990.

[8] A. Gafni, "Space Management and Cancellation Mechanisms for Time Warp," Technical Report TR-85-341, Univ. of Southern California, Los Angeles.

[9] F. Gomes, B.W. Unger, S. Franks, and J. Cleary, "Multiplexed State Saving for Bounded Rollback," *Proc. 1997 Winter Simulation Conf.,* pp. 460-467, Dec. 1997.

[10] D. Jefferson, "Virtual Time," *ACM Trans. Programming Languages and Systems,* vol. 7, no. 3, pp. 404-425, July 1985.

[11] Y.B. Lin and E.D. Lazowska, "Processor Scheduling for Time Warp Parallel Simulation," *Advances in Parallel and Distributed Simulation,* pp. 11-14, 1991.

[12] Y.B. Lin, B.R. Preiss, W.M. Loucks, and E.D. Lazowska, "Selecting the Checkpoint Interval in Time Warp Simulation," *Proc. Seventh Workshop Parallel and Distributed Systems (PADS '93),* pp. 3-10, May 1993.

[13] A.C. Palaniswamy and P.A. Wilsey, "Adaptive Checkpoint Intervals in an Optimistically Synchronized Parallel Digital System Simulator," *Proc. IFIP TC/WG10. Fifth Int'l. Conf. Very Large Scale Integration,* pp. 353-362, Sept. 1993.

[14] A.C. Palaniswamy and P.A. Wilsey, "An Analytical Comparison of Periodic Checkpointing and Incremental State Saving," *Proc. Seventh Workshop Parallel and Distributed Systems (PADS '93),* pp. 127-134, May 1993.

[15] B.R. Preiss, W.M. Loucks, and D. MacIntyre, "Effects of the Checkpoint Interval on Time and Space in Time Warp," *ACM Trans. Modeling and Computer Simulation,* vol. 4, no. 3, pp. 223-253, July 1994.

[16] F. Quaglia and V. Cortellessa, "Rollback-Based Parallel Discrete Event Simulation by Using Hybrid State Saving," *Proc. Ninth European Simulation Symp.,* pp. 275-279, Oct. 1997

[17] F. Quaglia, "Event History Based Sparse State Saving in Time Warp," *Proc. 12th Workshop Parallel and Distributed Systems (PADS '98),* pp. 72-79, May 1998.

[18] F. Quaglia, "Combining Periodic and Probabilistic Checkpointing in Optimistic Simulation," *Proc. 13th Workshop Parallel and Distributed Systems (PADS '99),* pp. 109-116, May 1999.

[19] F. Quaglia, "Fast-Software-Checkpointing in Optimistic Simulation: Embedding State Saving into the Event Routine Instructions," *Proc. 13th Workshop Parallel and Distributed Systems (PADS '99),* pp. 118-125, May 1999.

[20] R. Ronngren and R. Ayani, "Adaptive Checkpointing in Time Warp," *Proc. Eighth Workshop Parallel and Distributed Systems (PADS '94),* pp. 110-117, May 1994.

[21] R. Ronngren, M. Liljenstam, and J. Montagnat, "A Comparative Study of State Saving Mechanisms for Time Warp Synchronized Parallel Discrete Event Simulation," *Proc. 29th Ann. Simulation Symp.,* pp. 5-14, 1996.

[22] S. Skold and R. Ronngren, "Event Sensitive State Saving in Time Warp Parallel Discrete Event Simulations," *Proc. 1996 Winter Simulation Conf.,* Dec. 1996.

[23] H.M. Soliman and A.S. Elmaghraby, "An Analytical Model for Hybrid Checkpointing in Time Warp Distributed Simulation," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 10, pp. 947-951, Oct. 1998.

[24] J. Steinman, "Incremental State Saving in SPEEDES Using C Plus Plus", *Proc. 1993 Winter Simulation Conf.,* pp. 687-696, Dec. 1993.

[25] K. Umamageswaran, K. Subramani, P.A. Wilsey, and P. Alexander, "Formal Verification and Empirical Analysis of Rollback Relaxation," *J. Systems Architecture,* vol. 44, pp. 473-495, 1998.

[26] B.W. Unger, J. Cleary, A. Covington, and D. West, "External State Management System for Optimistic Parallel Simulation," *Proc. 1993 Winter Simulation Conf.,* pp. 750-755, Dec. 1993.

**Francesco Quaglia** received the Laurea degree in electronic engineering in 1995 and the PhD degree in computer engineering in 1999 from the University of Rome "La Sapienza." From summer 1999 to summer 2000, he held an appointment at CNR ("Consiglio Nazionale delle Ricerche," Italy). Currently, he is an assistant professor at the University of Rome "La Sapienza," His research interests include parallel discrete event simulation, parallel computing, fault-tolerant programming, and performance evaluation of software/hardware systems. He regularly serves as a referee for several international conferences and journals. He has served on the program committee of the 14th and 15th editions of the Workshop on Parallel and Distributed Simulation (PADS).