Combining Periodic and Probabilistic Checkpointing in Optimistic Simulation*

Francesco QUAGLIA
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
e-mail: quaglia@dis.uniroma1.it

Abstract

This paper presents a checkpointing scheme for optimistic simulation which is a mixed approach between periodic and probabilistic checkpointing. The latter, basing on statistical data collected during the simulation, aims at recording as checkpoints states of a logical process that have high probability to be restored due to rollback (this is done in order to make those states immediately available). The periodic part prevents performance degradation due to state reconstruction (coasting forward) cost whenever the collected statistics do not allow to identify states highly likely to be restored. Hence, this scheme can be seen as a highly general solution to tackle the checkpoint problem in optimistic simulation. A performance comparison with previous solutions is carried out through a simulation study of a store-and-forward communication network in a two-dimensional torus topology.

1 Introduction

Parallel discrete event simulation aims at speeding up the simulation process of complex systems through the use of multiple processors. Distinct parts of the simulated system are modeled by distinct interacting logical processes (LPs) [6]. An LP executes a sequence of events, and each event execution possibly schedules new events to be executed at later simulated time. LPs schedule events among each other by exchanging messages carrying the content and the occurrence time (timestamp) of the event. Simulation results are correct if each LP executes its events in non-decreasing timestamp order. In such a case the execution is said to satisfy *causality*. In order to guarantee an execution satisfying causality, LPs synchronize through an appropriate protocol.

In optimistic synchronization, also known as Time Warp [9, 10], events may be executed in violation of timestamp order as no "block until safe" policy is considered (i.e., an LP executes events unless its pending events set is empty); in such a case, causality is guaranteed by rollback which recovers the state of the LP back in simulated time to an error-free value. The principal advantage of this type of synchronization, over blocking based synchronization protocols, is that it offers the potential for greater exploitation of parallelism. The achievable speedup is, however, limited by the cost to support rollback and the cost of rollback itself.

One of the most significant costs in supporting rollback is the cost of checkpointing that guarantees the ability to reconstruct state information related to past values of the simulated time. The cost of checkpointing and the impact of the checkpointing scheme on the state recovery time are exactly what we study in this paper. In particular, we focus on the checkpointing scheme known as *sparse state saving*. In this scheme, only a subset of the states of the LP are recorded as checkpoints (this is done in order to keep short the CPU time spent recording state information), and state recovery to an unrecorded state is realized by reloading the latest checkpoint preceding that state and re-updating state variables through the re-play of intermediate events (coasting forward), thus adding to state recovery a time penalty due to state reconstruction.

We investigate on how the usage of statistical data related the evolution of the LP in simulated time can be used for the purpose of checkpointing intelligently. The term "intelligently" means that the checkpointing scheme should record as much states as possible from among those states that will be restored due to rollback (in order to keep short the state recovery time), and should record as few states as possible from among all the others (in order to keep low the checkpointing overhead). In other words, the scheme should optimize the positions of checkpoints in the simulated time in order to reduce the length of coasting forward.

An attempt to this kind of investigation has been presented in [16], where for the first time the progress of the simulated time is taken into account while selecting checkpoint positions. However the presented checkpointing scheme is based on the following assumption which has never been tested: "large simulated time increments are more likely than small ones to correspond to state transitions that will be undone in the future of the simulation". The scheme takes a checkpoint of the state of the LP whenever an event is being executed which determines a large simulated time increment.

We present in this paper a scheme that avoids previous assumption. We use a technique similar to the one presented in [3] in order to estimate the probability for a rollback to

*Work partially supported by CNR grant CTB96.02181.PS12.

occur in a given simulated time interval (delimited by timestamps of two consecutive events). Then, basing on the estimated probability value, a *probabilistic* decision is taken on whether to checkpoint or not the state of the LP corresponding to the starting point of the interval. The higher the estimated probability value, the higher the probability that the decision forces the LP to take a checkpoint.

Such a probabilistic approach is well suited for simulations in which statistical data clearly identify states more likely to be restored than others (i.e., simulated time intervals in which rollbacks occur with probability higher than in others). In the opposite case, statistical data no useful information provide for selecting good checkpoint positions, possibly leading probabilistic checkpointing to be ineffective. Hence, we derive a general checkpointing scheme, suited for any simulation, by combining previous probabilistic decision with periodic checkpointing. In our scheme, the more effective probabilistic checkpointing, the less frequently periodic checkpoints are taken. The adaptive tuning of the checkpoint period is based on the on-line monitoring of the real checkpointing-recovery overhead experienced by the LP, similarly to the approach adopted in the scheme in [4].

A simulation study of a store-and-forward communication network in a two-dimensional torus topology is presented for a performance comparison with previous solutions. The results show that, for the considered simulation problem, the proposed scheme achieves a reduction of the checkpointing-recovery overhead per event, leading to performance improvements.

The remainder of the paper is organized as follows. In Section 2 existing checkpointing schemes are briefly discussed. In Section 3 our solution is presented. Experimental results for a performance comparison with previous schemes are reported in Section 4.

2 Related Work

The goodness of a checkpointing scheme for optimistic simulation is usually evaluated in terms of: (i) CPU time spent recording state information, and (ii) efficiency of state recovery in case of rollback ([1]). Therefore when designing a checkpointing scheme, both aspects have to be taken into account. In the literature there exist three main checkpointing schemes, namely *copy*, *sparse* and *incremental* state saving, which exhibit different tradeoffs between previous aspects. They are shortly discussed in this section.

In copy state saving a checkpoint of the state of the LP is taken prior to the execution of each event [9] (usually such operation is realized via software). In this case, any state to be restored due to rollback is immediately available into a given state buffer, thus allowing short running time of the state recovery protocol[2]; however the checkpointing overhead may reach unacceptable levels, especially in the case of simulations with LPs having states of large granularity. For this kind of simulations, hardware solutions to accelerate the state saving operation can strongly improve performance [7], although they are unattractive primarily due to cost issues.

Sparse and incremental state saving aim at reducing the checkpointing overhead of copy state saving. Under sparse state saving the state of the LP is recorded according to some kind of periodicity (i.e., it is recorded either each χ event executions [11], χ being the checkpoint interval, or after T time units [1], or somewhere in the simulated time based on the event history [16]). Instead, under incremental state saving a log of before-images of the state variables modified during event execution is maintained (hence, at each event execution only the parts of the state that are going to be updated need to be recorded) [19, 22]. The disadvantage of both solutions consists of a time penalty added to state recovery. In the first case, the recovery to an unrecorded state is realized by reloading the latest checkpoint preceding that state and re-playing intermediate events in a coasting forward. In the second, state recovery is realized by copying back logged before-images into the state variables. Recently, schemes have been presented in [5, 15] which embed the incremental state saving mode on a sparse state saving basis in order to achieve better tradeoffs between the time spent recording state information and the state recovery time.

With regard to sparse state saving, that is the scheme under investigation in this paper, in all the existing solutions, except the one in [16], the checkpoint decision is taken without exploiting information about the simulated time evolution (i.e., checkpoints are taken randomly in the simulated time). The consequence is that the states within two successive checkpoints have the same probability to be restored due to rollback [4, 17]. Therefore the state recovery time is proportional to the distance, in terms of executed events, between successive checkpoints as coasting forward time is proportional to that distance. For the case of sparse state saving based on simulation events (i.e., each χ event executions), several analytical models have been proposed to determine the time-optimal value of the checkpoint interval. The models in [11, 13, 17] assume little variance of the event execution time. The model in [18] copes with simulations where previous assumption is not verified, hence generating a more general solution. Adaptive schemes for selecting the checkpoint interval have also been presented [4, 12, 17] in order to tackle both thrashing and throttling phenomena [14] and possible dynamic simulation conditions.

As outlined above, the only sparse state saving scheme that exploits information about the simulated time evolution (with the aim at optimizing the positions of checkpoints, instead of taking them randomly in the simulated time) has been described in [16]. In this scheme, the checkpoint decision is taken on the basis of the observation of differences between timestamps of successive 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. As pointed out in the Introduction, the as-

[1] Beyond performance, another factor determining the goodness of the scheme is the amount of memory used, however investigations on this aspect are out of the scope of this paper.

[2] At best state recovery involves only the setting of a pointer to the state buffer recording the state to be restored.

sumption that limits the generality of this scheme is: "large simulated time increments are more likely to correspond to over-optimistic[3] state transitions than small ones"([4]). The largeness of the increment (i.e., whether to checkpoint or not prior to a state transition) is defined by a threshold which is a function of the average simulated time increment due to the execution of an event.

The checkpointing scheme presented in this paper exploits information about the simulated time evolution (in order to optimize checkpoint positions) and avoids previous assumption, hence it is a more general solution to tackle the checkpoint problem in optimistic simulation.

3 The Checkpointing Scheme

In this section we first introduce the notion of probabilistic checkpointing. Then we show how it can be combined with periodic checkpointing. Finally, a method for collecting statistical data to estimate parameters needed in the probabilistic part and a technique to cope with non-stationary simulation conditions are presented.

3.1 Probabilistic Checkpointing

Let $\Delta_{LVT}(e)$ denote the Local Virtual Time (LVT) increment (i.e., the simulated time increment) at the LP due to the execution of the event e. This quantity is evaluated as the difference between the timestamp of e, denoted as $ts(e)$, and the LVT of the LP just before the execution of e, denoted as $LVT_{prior(e)}$. To each event e is associated a simulated time interval, namely $I(e)$, whose length is $\Delta_{LVT}(e)$, defined as follows:

$$I(e) = (LVT_{prior(e)}, ts(e)] \quad (1)$$

We denote as $P_r(\Delta_{LVT}(e))$ the probability for a rollback to occur in the simulated time interval $I(e)$ (a rollback in that interval occurs either because events are scheduled later with timestamps in that interval or because the sender of the message that schedules e rolls back undoing e). If the value of $P_r(\Delta_{LVT}(e))$ is *high*, then in all likelihood there will be a rollback in the interval $I(e)$, and the state of the LP immediately preceding e's execution will have to be restored due to rollback. On the contrary, if that value is small the state of the LP immediately preceding e's execution is extremely unlikely to be restored.

The probabilistic part of the checkpointing scheme uses $P_r(\Delta_{LVT}(e))$ to take the checkpoint decision *probabilistically*, in the sense that a checkpoint is taken with a certain probability before e's execution ([5]). The probabilistic decision is based on the following observation: the value $P_r(\Delta_{LVT}(e))$ represents the likelihood to have an over-optimistic state transition due to the execution of the event e. Hence, the value $1 - P_r(\Delta_{LVT}(e))$ can be seen as the likelihood for the event e to not determine that *bad* transition. This means that, theoretically, a fraction $1 - P_r(\Delta_{LVT}(e))$ of the event e determines a *safe* transition that should be accepted without taking a checkpoint before e's execution. The remaining fraction $P_r(\Delta_{LVT}(e))$ determines an *unsafe* transition, that should be accepted taking a checkpoint before e's execution.

By previous observation, the probabilistic part of the checkpointing scheme takes the checkpoint decision as follows: a value α uniformly distributed between 0 and 1 is extracted and a checkpoint is taken before the execution of e if $\alpha > (1 - P_r(\Delta_{LVT}(e)))$. The following expression synthesizes such a probabilistic rule:

Probabilistic-Rule:
if $(\alpha = rand()) > (1 - P_r(\Delta_{LVT}(e)))$
then take a checkpoint before the execution of e

According to the probabilistic decision, the higher the probability for an event e to determine an over-optimistic state transition (i.e., the higher the probability that a rollback will occur in the simulated time interval $I(e)$), the higher the probability to have the state prior to that transition recorded as a checkpoint.

3.2 Combining Probabilistic and Periodic Checkpointing

The probabilistic checkpointing rule described in previous section does not always guarantee good performance (i.e., a good tradeoff between time spent recording state information and state recovery time). This may happen, for example, whenever for any event e the probability $P(\Delta_{LVT}(e))$ of rollback occurrence in the simulated time interval $I(e)$ is quite low. In such a case, the probabilistic decision extremely unlikely pushes the LP to take a checkpoint (i.e., very few states are recorded as checkpoints in the course of the simulation). This is a highly undesirable behavior as, in case of rollback, it might induce long state recovery time due to long coasting forward.

In order to design a general checkpointing scheme, suitable either for the case of probability values that give useful information to optimize checkpoint positions or not, we combine the probabilistic decision together with periodic checkpointing (i.e., checkpointing based on the number of executed events). For this purpose, each LP is endowed with two integer variables, namely $interval$ and $event\_ex$ respectively. The variable $interval$ represents the current value of the checkpoint interval of the LP, that is the maximum number of event executions allowed between two successive checkpoint operations. The variable $event\_ex$ represents the actual distance, in terms of events, from the last checkpoint operation.

The mixing between probabilistic and periodic checkpointing is realized as follows: upon the execution of e, a

[3] We use the term "over-optimistic" to denote a state transition that will be undone in the future of the simulation.

[4] Although this assumption is suited for many synthetic workloads [3, 16], it has never been tested for real-world simulation models.

[5] A probabilistic decision has been already used in [2] in the context of throttled Time Warp. In that paper, a protocol is introduced that probabilistically delays the execution of an event depending on statistics related to the differences between timestamps of scheduled events and on forecasts of timestamps of forthcoming events.

checkpoint of the state of the LP is taken either if the predicate of the probabilistic rule is evaluated to $TRUE$ or if $event\_ex = interval$. The following expression synthesizes such a mixed rule:

Mixed-Rule:
if $((\alpha = rand()) > (1 - P_r(\Delta_{LVT}(e)))$
or $(event\_ex = interval)$
then take a checkpoint before the execution of e

The probabilistic part tries to optimize checkpoint positions in the simulated time (i.e., it probabilistically forces the LP to record states that have high probability to be restored). The periodic part prevents unbounded state recovery time due to coasting forward whenever the probabilistic part does not allow a good selection of checkpoint positions. The value of $interval$ should be selected basing on the effectiveness of the probabilistic decision. In particular, the more effective probabilistic checkpointing, the higher the value of $interval$. In the next section a method to calculate suitable values for $interval$ is presented.

3.3 Calculating the Checkpoint Interval

We base the calculation of $interval$ on a monitoring technique similar to the one presented in [4]. We consider the execution of the LP as divided into observation periods. Each period consists of N committed/rolled-back events. The value of N should be chosen in order to ensure statistical data collected in any period to be meaningful (suggestions to solve this problem have been already pointed out in other studies [17]). All the observation periods are numbered, starting from one. At the beginning of the n-th period, a value for $interval$ is selected, denoted $interval_n$. Hence, during the n-th period, checkpoints are taken by using the previous **Mixed-Rule** applied considering the selected value for $interval_n$.

The dynamic recalculation of the value of $interval_n$ is realized as follows. A checkpointing-recovery cost function COF is continuously monitored by the LP (similarly to what happens in the schemes in [4, 16]). Denoting with SS_n the CPU time spent by the LP for state saving operations during the n-th period and with CF_n the CPU time spent for coasting forward operations in the same period, then the value of the cost function at the n-th period is computed as $COF_n = SS_n + CF_n$.

At the beginning of the first period, the value $interval_1 = 1$ is chosen, hence, according to the **Mixed-Rule**, the state of the LP is recorded before the execution of each event (the recording of the state before the execution of each event during the first observation period is a feature common to several existing adaptive checkpointing schemes [4, 16]).

The value of $interval$ is increased by one at the beginning of each of the successive observation periods until COF shows an increase. In this case, the adaptation direction of $interval$ is inverted (it is decreased by one). The inversion of the adaptation direction takes place each time the value of COF observed in the last period is greater than the one in the previous period.

```
PROCEDURE select interval_n :
  if n = 0 then interval_n = 1;
  if n = 1 then {interval_n = interval_{n-1} + 1; up = TRUE};
  if n > 1 then
        if up
        then
          if (COF_{n-2} < COF_{n-1})
          then {interval_n = max(1, interval_{n-1} - 1);
                up = FALSE}
          else interval_n = min(max_interval, interval_{n-1} + 1)
        else
          if (COF_{n-2} < COF_{n-1})
          then {interval_n = min(max_interval, interval_{n-1} + 1);
                up = TRUE}
          else interval_n = max(1, interval_{n-1} - 1)
```

Figure 1. The Selection Algorithm.

The proposed recalculation aims at increasing the value of $interval$ in order to induce the LP to take the checkpoint decision basing only on the probabilistic part of the **Mixed-Rule**. If probabilistic checkpointing is effective, i.e., the time spent due to coasting forward is kept short (this is monitored through the observation of the value of the cost function), then periodic checkpointing is useless thus the value of $interval$ is further increased. If probabilistic checkpointing is not effective (thus the LP experiences long coasting forwards which determine a performance decrease, i.e., an increase in the COF function), $interval$ is decreased until the checkpointing overhead due to both probabilistic and periodic checkpointing determines, in turn, a performance decrease (i.e., an increase in the COF function). An upper limit $max\_interval$ on the value of $interval$ is used because of the LP memory consumption problem[6]. In the experiments presented in Section 4 we set $max\_interval = 30$ (the same value has been selected in the scheme in [4]).

In Figure 1 the algorithm for selecting $interval_n$ is shown. A boolean variable up is used to indicate whether the current adaptation direction of the checkpoint interval is towards increasing values or not.

3.4 Estimating $P(\Delta_{LVT}(e))$

The probabilistic part of the **Mixed-Rule** requires, for each event e, the knowledge of the probability $P(\Delta_{LVT}(e))$. In order to estimate such value, we use a scheme similar to the one presented in [3]. We define simulated time points t_i such that $t_i = 0$ if $i = 0$ and $t_i < t_{i+1}$. Then we decompose the positive simulated time semi-axis into intervals $I_i = [t_i, t_{i+1})$. For each event e, there exists an interval I_i such that $t_i \leq \Delta_{LVT}(e) < t_{i+1}$ (i.e., the simulated time increment due to the execution of e falls into the interval I_i).

For each interval I_i, the amount of processes events

[6] Since rollback to Global-Virtual-Time (GVT) is possible, upon the recovering of the storage allocated for obsolete states and messages, at least one state older than GVT and also all the input messages with timestamps larger than the simulated time of that state need to be retained. So if very few states are recorded, then a large amount of input messages cannot be garbage collected giving rise to an increase in the frequency of storage recovering (due to frequent saturation of the input queue), thus making performance worse.

that generate simulated time increment $\Delta_{LVT}(e)$ of length within that interval is counted (namely N_i). Furthermore, for each interval I_i, the amount of rollback occurrences in simulated time intervals $I(e)$, such that $t_i \leq \Delta_{LVT}(e) < t_{i+1}$, is counted (namely R_i). Then, for an event e such that $t_i \leq \Delta_{LVT}(e) < t_{i+1}$, $P(\Delta_{LVT}(e))$ is approximated as R_i/N_i (by convention, if $N_i = 0$, we assign the value 0 to that ratio).

Several different techniques can be used to generate previous decomposition. A simple uniform decomposition limited on the right can be obtained by imposing length equal to δ on each interval I_i and by defining a value t_{max} such that $I_{max} = [t_{max}, +\infty)$. In this case the value of max determines the number of intervals of the decomposition (i.e., the amount of memory destined to previous counters N_i and R_i). Furthermore, the value of δ can be chosen depending on the average values of the distribution functions of the timestamp increment. In this approach, the individuation of the counters to be updated each time an event is executed or upon the occurrence of a rollback, or the individuation of counters to estimate probability values is quite simple and introduces negligible overhead. For example, when an event e is executed, the index i of the counter N_i to be updated is easily computed as

$$i = \begin{cases} \lfloor \frac{\Delta_{LVT}(e)}{\delta} \rfloor & \text{if } \Delta_{LVT}(e) < t_{max} \\ max & \text{if } \Delta_{LVT}(e) \geq t_{max} \end{cases} \qquad (2)$$

Although very simple, the uniform decomposition usually does not allow to obtain uniform density of observations in all the intervals as events that generate small simulated time increment are more likely than events generating simulated time increment much larger than zero. This problem can be solved by using non-uniform decompositions of the simulated time positive semi-axis, such as, for example, the logarithmically spaced one used in [3]. As the study on the optimality of a decomposition for a given simulation setting is out of the aims of this paper, the experimental results reported in Section 4 have been obtained by using a simple uniform decomposition (the parameters of the decomposition are reported in the same section).

3.5 Coping with non-Stationary Behavior

As explained in previous subsection, to estimate the probability of rollback occurrence in a simulated time interval $I(e)$ of length $\Delta_{LVT}(e)$ within the interval $I_i = [t_i, t_{i+1})$, we use counters N_i and R_i which record, respectively, the amount of executed events that generate simulated time increment in I_i and the amount of rollback occurrences in simulated time intervals of length L such that $t_i \leq L < t_{i+1}$.

In this subsection we propose a solution to cope with non-stationary behavior of the simulation using duplication of previous counters. The aim is to guarantee a *good* estimate of probability values of rollback occurrences during the lifetime of the simulation. Note that the solution we propose copes with non-stationary behavior in which probability values can be assumed to be stationary over a non-minimal number of consecutive events (where non-minimal number means several observation periods). Hence, non-stationarity means that the simulation follows a given (stationary) behavior along a number ℓ of observation periods, then it starts to follow a different behavior for ℓ' observation periods and so on[7].

In order to cope with this type of non-stationarity, we assume that for each counter N_i (resp. R_i) there exists a duplicated counter N_i^d (resp. R_i^d). The duplicated counter N_i^d (resp. R_i^d) is updated whenever the original counter N_i (resp. R_i) is updated. However, at the beginning of each observation period both N_i^d and R_i^d are set to zero. This means that, at the end of the n-th observation period, N_i^d counts the amount of events generating simulated time increments of length within I_i during the n-th period. Similarly, R_i^d counts the amount of rollbacks occurred during the n-th period in simulated time intervals of length L such that $t_i \leq L < t_{i+1}$. Hence, at the end of the n-th period the ratio R_i^d/N_i^d is an estimate of the probability $P(\Delta_{LVT}(e))$ (for an event e such that $t_i \leq \Delta_{LVT}(e) < t_{i+1}$) which takes into account statistics related to the n-th period only (also in this case we adopt the convention to assign the value 0 to that ratio if $N_i^d = 0$).

At the end of each observation period duplicated counters are used to update the original counters as follows: a value γ is chosen such that $0 \leq \gamma \leq 1$; if $R_i^d/N_i^d < (1-\gamma)R_i/N_i$ or $R_i^d/N_i^d > (1+\gamma)R_i/N_i$, then N_i^d is copied onto N_i and R_i^d is copied onto R_i. Such updating rule means: if the estimate based on statistics related only to the last observation period is enough different from the estimate based also on previous statistics, then the simulation is supposed to have changed its behavior. Therefore, previous statistics are discarded. The value of γ determines the difference between the estimates which must be considered as representative of a change in the simulation behavior. For the particular simulation problem considered in Section 4, we noted that values between 0.05 and 0.15 allows to obtain the best performance results.

4 A Performance Comparison

In this section experimental results are reported to compare the performance achievable by using the checkpointing scheme proposed in this paper (hereafter S1) to the one obtained by using Ronngren and Ayani's scheme (S2) [17], Fleischmann and Wilsey's scheme (S3) [4] and the scheme by Quaglia (S4) [16]. Both S2 and S3 take the checkpoint decision only on the basis of simulation events (i.e., without exploiting information on the simulated time evolution). The S2 scheme is based on an analytical model of the LP execution time which defines the value of the time-optimal checkpoint interval under the assumption that

[7] Shocks in the behavior of the simulation are not tackled by our solution. Those shocks actually produce situations in which the selection of checkpoint positions based on collected statistics might result ineffective. In such a case good performance are guaranteed by periodic checkpoints taken by the proposed **Mixed-Rule**.

there is no correlation between where in simulated time rollbacks occur and the timestamp of events that are checkpointed. The model is used to recalculate the value of the checkpoint interval basing on the variations of the rollback frequency. The S3 scheme selects an initial value and an adaptation direction for the checkpoint interval (the adaptation step is 1). Then, the adaptation direction is inverted each time the monitored checkpointing-recovery overhead experienced by the LP shows a significant increase. The S4 scheme, which has already been discussed in Section 2, takes the checkpoint decision basing on the observation of differences between timestamps of consecutive events. Before showing the results of the comparative analysis, the used simulator and the selected benchmark are described.

4.1 The Simulator and the Benchmark

As hardware architecture we used a cluster of machines (Pentium 120 MHz - 32 Mbytes RAM) connected via Ethernet. The number of machines in the cluster is eight. Interprocessor communication relies on message passing supported by PVM [21]. There is an instance of the Time Warp kernel on each processor. The kernel schedules LPs for event execution according to the Smallest-Timestamp-First policy; antievents are sent aggressively (i.e., as soon as the LP rolls back [8]); "fossil collection" is executed on demand.

We propose simulation results of a communication network in a two-dimensional torus configuration. Messages are transmitted according to the store-and-forward policy and the x-y-routing algorithm [20]. We consider a fixed message population (10 messages for each node), hence, when a message reaches the destination, another message is reinserted into the network. Message lengths are selected from an uniform distribution (between 10 bytes and 3 Kbytes) and the message transmission time is proportional to the message length (1×10^{-2} unit times per byte). We assume there are two unidirectional links between neighboring nodes (one link for each direction). As only one message can be transmitted at a time, there may be queuing delays at links (the queuing discipline is FCFS). Hence, a queuing server is associated with each link. The service time is equal to the message transmission time. Service times are precomputed and messages are immediately forwarded to the next node. The dimension of the network is 4×4. Each node is modeled by a single LP.

There are two hot spot nodes to which 20% of all messages are destined (the remaining portion of the messages are uniformly destined to the other nodes). The hot spots change during the lifetime of the simulation in order to obtain non-stationarity in the behavior of the simulation.

The size of the state of an LP is 1 Kbyte (the corresponding state saving time is around 35 microseconds), the event routine time is around 80 microseconds. These values outline that the considered simulation model represents an application with medium state granularity with respect to the event routine time. This model is therefore suited for studying sparse state saving schemes[8].

[8] In case of small state granularity, copy state saving usually represents the best solution. Instead, in case of large state granularity, incremental state saving usually outperforms the other schemes. So, small and large state granularity models push out of the application field of sparse state saving.

The experiments were performed with 2, 4 and 8 processors; the obvious assignment of LPs onto processors has been adopted. The observation period was fixed at 500 events for all the checkpointing schemes. Furthermore, for the S1 scheme, we adopted an uniform decomposition of the positive simulated time semi-axis limited on the right, with $\delta = 10.0$ time units and $t_{max} = 100.0$ time units. Furthermore, the value 0.1 was chosen for γ.

We report measures related to: the average distance between checkpoints (i.e., the average checkpoint interval), the average coasting forward length, the efficiency (i.e., the ratio between the total number of committed events and the total number of executed events excluding coasting forward ones) and the event rate (i.e., committed events per second). We base our comparison on the average value analysis. All parameter values result as the average of 10 runs (each run contains at least 800000 committed events) ([9]).

4.2 Experimental Results

The obtained results are reported in Table 1, Table 2 and Table 3, for the case of 2, 4 and 8 processors respectively. In all the cases, the highest values of the average checkpoint interval are obtained under S2. The average checkpoint intervals of S1, S3 and S4 are quite similar (the maximum distance is around 4% and is noted in the case of simulations run on 4 processors).

Regarding the average coasting forward length, the largest values are obtained under S2, whereas the smallest values are usually obtained under S1 and S4. Combining results related to the average checkpoint interval and the average coasting forward length, a basic observation can be deduced. For both the schemes S2 and S3 we get that the average coasting forward length is around half of the checkpoint interval minus 0.5 (as both S2 and S3 take checkpoints randomly in the simulated time, the length of coasting forward is uniformly distributed between zero and the average checkpoint interval minus one [4, 17]). For S1 and S4 previous relation is not satisfied (i.e., under these schemes the average coasting forward is actually less than half of the checkpoint interval minus 0.5). Regarding S1, this means that the probabilistic part of the checkpointing scheme actually allows a good placing of checkpoints in the simulated time with respect to the rollback points. Compared to S3, both S1 and S4 allow shorter state recovery time with around the same (or even larger) average distance between checkpoints, with the advantage of a reduction of the checkpointing-recovery overhead per event.

The efficiency values of the checkpointing schemes are quite similar, meaning that the schemes themselves affect the percentage of CPU time spent executing events that are not rolled back almost in the same way. In general, S2 is more efficient than all the other schemes, and both S1 and S4 are usually slightly less efficient than S3. This phe-

[9] The ten runs were all done with different random seeds.

Table 1. Results with 2 Processors.

Checkpointing Scheme	Average Checkpoint Int.	Average Coast. Forw.	Efficiency [%]	Event Rate
S1	3.65	1.21	90.11	9,199
S2	4.15	1.56	91.24	8,287
S3	3.59	1,28	90.47	8,890
S4	3.61	1.22	89.85	8,981

Table 2. Results with 4 Processors.

Checkpointing Scheme	Average Checkpoint Int.	Average Coast. Forw.	Efficiency [%]	Event Rate
S1	3.32	1.03	83.32	14,971
S2	3.80	1.41	84.35	13,839
S3	3.19	1.09	83.98	14,336
S4	3.29	1.06	83.55	14,557

nomenon is supposed to derive from the longer state recovery time (i.e., the longer average coasting forward) of S2 originating a throttled execution.

The overall effects of previous parameter values is quantified by results of the event rate. S1 and S4 usually show better event rate due to both their short coasting forward and their low checkpointing overhead.

Obtained data show that S1 and S4 outperform the other schemes, and that S1 usually shows event rate better than S4. Latter behavior can be explained by looking at data reported in Figure 2 and in Figure 3 related, respectively, to the frequency of rollback occurrence in the intervals of the decomposition of the simulated time semi-axis adopted in the S1 scheme and to the frequency of event execution generating simulated time increment in those intervals. Data in Figure 2 show that independently of the number of processors, rollbacks are actually unlikely to occur in simulated time intervals corresponding to short simulated time increments (i.e., up to 30 simulated time units), whereas they are highly likely to occur in simulated time intervals corresponding to large simulated time increments. This would lead to the conclusion that, performance of S4 should be better than that of S1, as the assumption underlying the S4 scheme is actually verified. However, looking at data reported in Figure 3 we get that the majority of event executions generate small simulated time increments (i.e., up to 30 simulated time units), thus, the majority of rollbacks occur in intervals corresponding to small simulated time increments. Therefore many states that have to be restored are actually not recorded in the S4 scheme. On the contrary, the randomization introduced by the probabilistic part of S1 allows sometimes those states to be recorded (with a reduction of the average coasting forward length).

Table 3. Results with 8 Processors.

Checkpointing Scheme	Average Checkpoint Int.	Average Coast. Forw.	Efficiency [%]	Event Rate
S1	2.50	0.65	72.06	24,202
S2	3.15	1.06	73.82	22,315
S3	2.52	0.75	72.56	22,609
S4	2.48	0.68	72.15	23,440

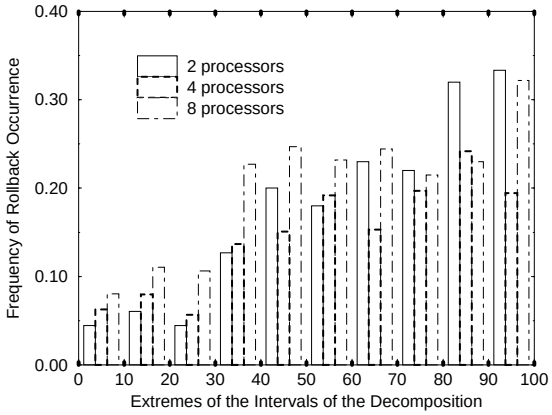

Figure 2. Frequency of Rollback Occurrence.

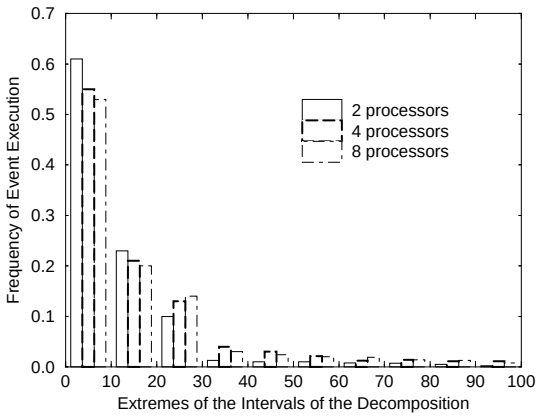

Figure 3. Frequency of Event Execution.

This outlines that S4 is suited for all the cases in which large simulated time increments are likely to correspond to over-optimistic state transitions and a non-negligible percentage of event executions generate large simulated time increments (so the number of rollbacks occurring in intervals corresponding to large simulated time increments is non-minimal and the scheme actually produces good checkpoint positions). For any other situation, the S1 scheme would represent a better solution.

Furthermore, note that it is unlikely to know prior to the execution of the simulation if the simulation itself shows the features which allow S4 to ensure good performance. Therefore, S1 may be preferable to other solutions in any case in which there is no a priori knowledge about the behavior of the simulation.

5 Summary

In this paper a checkpointing scheme for optimistic (Time Warp) simulators is presented. The scheme is based on a mixed approach between periodic and *probabilistic* checkpointing. The probabilistic part of the scheme aims at establishing a relation between checkpoints and rollback points in order to reduce the average rollback cost (i.e., it aims at optimizing checkpoint positions in the simulated time). To this purpose statistics on the simulated time evolution are collected for estimating the probability of rollback occurrence in a given simulated time interval. Then, a probabilistic decision on whether to checkpoint the state of the logical process corresponding to the starting point of the interval is taken. The probabilistic decision usually forces the logical process to take a checkpoint whenever the estimated probability of rollback occurrence in that interval is high. By mixing previous decision with periodic checkpointing we obtained a highly general checkpointing scheme, suited to any simulation scenario. We tested performance of the new scheme through simulations of a store-and-forward communication network. The obtained results have shown a performance improvement obtained by the new scheme through a reduction of the checkpointing-recovery overhead per event.

Acknowledgments. The author would like to thank Vittorio Cortellessa for his help in the preparation of the simulation code.

References

[1] S. Bellenot, "State Skipping Performance with the Time Warp Operating System", *Proc. of 6-th Workshop on Parallel and Distributed Simulation*, pp.33-42, January 1992.

[2] A. Ferscha, "Probabilistic Adaptive Direct Optimism Control in Time Warp", *Proc. of 9-th Workshop on Parallel and Distributed Simulation*, pp.120-129, June 1995.

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

[4] J. Fleischmann and P.A. Wilsey, "Comparative Analysis of Periodic State Saving Techniques in Time Warp Simulators", *Proc. of 9-th Workshop on Parallel and Distributed Simulation*, pp.50-58, June 1995.

[5] S. Franks, F. Gomes, B. Unger and J. Cleary, "State Saving for Interactive Optimistic Simulation", *Proc. of 11-th Workshop on Parallel and Distributed Simulation*, pp.72-79, June 1997.

[6] R.M. Fujimoto, "Parallel Discrete Event Simulation", *Communications of ACM*, Vol.33, No.10, 1990, pp.30-53.

[7] R.M. Fujimoto and G.C. Gopalakrishnan, "Design and Evaluation of the Rollback Chip: Special Purpose Hardware for Time Warp", *IEEE Transactions on Computers*, Vol.41, No.1, 1992, pp.68-82.

[8] A. Gafni, "Space Management and Cancellation Mechanisms for Time Warp", *Tech. Rep. TR-85-341*, University of Southern California, Los Angeles (Ca,USA).

[9] D. Jefferson and H. Sowizral, "Fast Concurrent Simulation Using the Time Warp Mechanism; Part I: Local Control", *Tech. Rep. N1906AF*, RAND Corporation, December 1982.

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

[11] Y.B. Lin, B.R. Preiss, W.M. Loucks and E.D. Lazowska, "Selecting the Checkpoint Interval in Time Warp Simulation", *Proc. of 7-th Workshop on Parallel and Distributed Simulation*, pp.3-10, May 1993.

[12] A.C. Palaniswamy and P.A. Wilsey, "Adaptive Checkpoint Intervals in an Optimistically Synchronized Parallel Digital System Simulator", *Proc. of IFIP TC/WG10.5 Int. Conf. on Very Large Scale Integration*, pp.353-362, September 1993.

[13] A.C. Palaniswamy and P.A. Wilsey, "An Analytical Comparison of Periodic Checkpointing and Incremental State Saving", *Proc. of 7-th Workshop on Parallel and Distributed Simulation*, pp.127-134, May 1993.

[14] B.R. Preiss, W.M. Loucks and D. MAcIntyre, "Effects of the Checkpoint Interval on Time and Space in Time Warp", *ACM Transactions on Modeling and Computer Simulation*, Vol.4, No.3, 1994, pp.223-253.

[15] F. Quaglia and V. Cortellessa, "Rollback-Based Parallel Discrete Event Simulation by Using Hybrid State Saving", *Proc. of 9-th European Simulation Symposium*, pp.275-279, October 1997.

[16] F. Quaglia, "Event History Based Sparse State Saving in Time Warp", *Proc. of 12-th Workshop on Parallel and Distributed Simulation*, pp.72-79, May 1998.

[17] R. Ronngren and R. Ayani, "Adaptive Checkpointing in Time Warp", *Proc. of 8-th Workshop on Parallel and Distributed Simulation*, pp.110-117, May 1994.

[18] S. Skold and R. Ronngren, "Event Sensitive State Saving in Time Warp Parallel Discrete Event Simulations", *Proc. of 1996 Winter Simulation Conference*, December 1996.

[19] J. Steinman, "Incremental State Saving in SPEEDES Using C Plus Plus", *Proc. of 1993 Winter Simulation Conference*, pp.687-696, December 1993.

[20] H. Sullivan and T.R. Bashkow, "A Large Scale, Homogeneous, Fully Distributed Parallel Machine", *Proc. of 4-th International Symposium on Computer Architecture*, March 1977.

[21] V.S. Sunderam, "A Framework for Parallel Distributed Computing", *Concurrency: Practice and Experience*, Vol.2, No.4, 1990.

[22] B.W. Unger, J.G. Cleary, A. Covington and D. West, "External State Management System for Optimistic Parallel Simulation", *Proc. of 1993 Winter Simulation Conference*, pp.750-755, December 1993.