Event History Based Sparse State Saving in Time Warp*

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 sparse state saving scheme for Time Warp parallel discrete event simulation. The scheme bases the selection of the states to be recorded on the event history of the logical processes. To this purpose, statistics on the virtual time advancement of the processes are collected for the prediction of virtual time intervals that are likely to contain rollback points; the states corresponding to the starting point of those intervals are recorded as checkpoints in order to reduce the average coasting forward. The percentage of states to be recorded is defined by a parameter whose value is dynamically recalculated on the basis of the on-line observation of the variation of a checkpointing-rollback cost function. Simulation results of synthetic workloads are presented for a performance comparison with previous schemes.

1 Introduction

In parallel discrete event simulation, concurrent logical processes (LPs) model the parts of the system to be simulated [7]. The interaction between LPs takes place through the exchange of *timestamped* messages, and the scheduling of an event for an LP at virtual time t is realized by sending to the LP a message with timestamp t. An underlying synchronization protocol guarantees *causality* by ensuring that each LP processes input messages in non-decreasing timestamp order.

The Time Warp synchronization protocol [11, 12] lets the LPs schedule simulation events whenever they are available (by optimistically assuming that the schedule does not violate causality) and uses rollback to recover from out of order computations. In particular, when an LP detects an out of order computation at virtual time t, it rolls back to its state immediately prior t, and resumes execution from that state.

*Work partially supported by grant No. ERB4050PL932483 from the Scientific Cooperation Network of the European Community "OLOS".

In order to guarantee the reconstructability of past states, Time Warp simulators are provided with well known checkpointing schemes such as *copy*, *sparse* and *incremental* state saving (recently schemes that mix features of both sparse and incremental approaches have been proposed in [6, 18]).

When copy state saving is adopted, the state of an LP is recorded as a checkpoint before processing each event [11]. In this case, a state to be restored due to rollback is always available, but, unless special hardware is employed to accelerate the state saving operation [9], the checkpointing overhead can reach unacceptable levels. Sparse and incremental state saving aim at reducing such an overhead. The first scheme records as checkpoints only a subset of states [1, 13]. The latter records the inverses of all the incremental changes of an LP state [22, 23]. The cost incurred in both solutions is a time penalty added to rollback. In the first case, the reconstruction of an unrecorded state (*missing* state) is realized by starting from a previous recorded one and by re-processing intermediate events (*coasting forward*). In the second, the state to be restored is regenerated by starting from the current LP state and by backward applying inverse changes.

Although each scheme is suitable for a given class of simulations[1], the most interesting, from a performance study point of view, is sparse state saving, due to the multiplicity of approaches that can be adopted for the selection of the states to be recorded. To the best of our knowledge, two approaches have been envisaged. One [1] records the state of the LP after it consumes a save period amount of CPU time. The other [13], also known as *periodic* state saving, selects the states to be recorded on the basis of simulation events (the state of an LP is periodically recorded each χ event executions, χ being the *checkpoint interval* of the LP). Both solutions do not take the event history of the LP into account. As a consequence, no correlation comes out between checkpoints and rollback points, thus lead-

[1] In [15] it has been proved that incremental state saving improves performance over the other schemes when both the fraction of the state updated at each event execution and the rollback length are minimal.

ing to a rollback cost proportional to the average distance between checkpoints. In these approaches the tradeoff between the checkpointing frequency and the checkpointing-rollback overhead has been the object of several studies [1, 13, 15, 16, 19, 20]. Furthermore, in the case of periodic state saving, adaptive techniques have been introduced to dynamically recalculate the value of the checkpoint interval. In [14, 17, 19] the recalculation is based on the variation of the rollback behavior (i.e., rollback frequency and rollback length) of the LP; in [5] it is based on the variation of the time spent due to checkpointing and coasting forward operations.

As an alternative approach, this paper introduces a sparse state saving scheme which bases the selection of the states to be recorded on the event history of the LPs. Intuitively, smaller rollback cost, compared to previous solutions, can be obtained if checkpoints are not taken randomly in virtual time, but correlated to the rollback points of the LPs. In order to achieve this goal, each LP observes how the scheduling of events affects its virtual time progression and, by keeping track of statistics related to such progression, tries to predict virtual time intervals (defined by timestamps of successive events) that are likely to contain rollback points. The states corresponding to the starting point of those intervals are recorded as checkpoints in order to reduce the average coasting forward.

Our approach is based also on the observation of a checkpointing-rollback cost function (actually this function is the same as the one presented in [5]) which reflects the tradeoff between the number of states recorded and the goodness of the prediction of the states that have to be restored. The on-line variation of such a function is used for dynamically recalculating the value of a parameter which defines the percentage of states to be recorded.

The exploitation of the event history joins this sparse state saving approach and several techniques introduced with the aim of limiting the optimism of Time Warp (for example those in [3, 4, 21]), with the difference that the reduction of the rollback overhead is achieved by reducing the rollback cost instead of the rollback frequency.

An experimental study of synthetic workloads is presented for a performance comparison with the adaptive methods in [19] and in [5]. The results show that the new approach achieves a reduction of the average coasting forward which leads to a significant performance improvement.

The remainder of the paper is organized as follows. In Section 2 some details about sparse state saving are reported; furthermore, the adaptive solutions presented in [19] and in [5] are summarized. In Section 3 the sparse state saving scheme is described. Experimental results are reported in Section 4. Conclusions, a discussion on limitations of our scheme and future work constitute Section 5.

2 Background

Checkpointing has two distinct effects on a Time Warp simulation. First, some processing time is spent running the state saving protocol; second, memory is consumed to record the state (or part of it) of an LP. Furthermore, the checkpointing scheme has a direct impact on the rollback cost. This paper focuses on the effect of sparse state saving on the execution time of the simulation.

As outlined in the Introduction, the sparse state saving schemes proposed in literature select the states to be recorded without exploiting information about the event history of the LPs. Several studies [5, 19] proved that by using this approach there is no correlation between where in virtual time rollbacks occur and the timestamp of events that are checkpointed. As a consequence, the average length of coasting forward (i.e., the average number of events to be re-executed upon the regeneration of a missing state) results proportional to the average distance (in terms of executed events) between successive state saving operations. From the above considerations, two opposite effects appear while decreasing the checkpointing frequency of an LP: (i) the CPU time spent for state saving decreases, (ii) the average cost of rollback increases.

In the case of sparse state saving based on real CPU execution time, it has been shown [1] that, for unsaturated workloads, the best execution times are obtained when the state is recorded after a save period amount of CPU execution time equal to zero. This is because the cost of checkpointing in terms of CPU time can be ignored if a processor has no productive simulation work to be performed.

In the case of periodic state saving, several analytical models have been proposed to describe the simulation execution time of an LP as a function of its checkpoint interval [13, 15, 19]. Among those models, particularly interesting is the one in [20], which takes into account how the granularities of different event types affect the average regeneration time of a missing state.

The extended experimental study in [16] pointed out that, under atomic coasting forward, the variation of the checkpoint interval strongly affects the rollback behavior of the LPs. Presented results showed that when the checkpoint interval slightly increases a *throttling* effect appears which tends to reduce the number of rollbacks; when the checkpoint interval is largely increased (i.e., in the case of long average coasting forward), a *thrashing* effect gives rise to an increase in the number of rollbacks.

Due to the presence of these phenomena and to the proper dynamism of the rollback behavior of the LPs, the selection of a fixed value of the checkpoint interval usually results as being unsuitable for providing performance optimization. Hence, several authors have introduced adaptive periodic state saving techniques.

Most of them [5, 17, 19] are based on the observation of some parameters of the LP behavior over a certain number of scheduled events, referred to as *observation period*, and recalculate the checkpoint interval at the beginning of each period. A different approach can be found in [14], where the recalculation is executed every *Global-Virtual-Time* (GVT) evaluation[2].

Performance achievable by using the checkpointing scheme presented in this paper is compared in Section 4 to the one obtained by using the adaptive solutions proposed by Ronngren and Ayani, and by Fleischmann and Wilsey. Both these solutions are sketched below.

Ronngren and Ayani's Approach. The adaptive checkpointing algorithm presented by Ronngren and Ayani [19] is based on an analytical model of the LP execution time. The model lies on the following basic assumptions: (i) non preemptive rollback, (ii) atomic rollback procedure.

Once measured the parameters k_{obs} (number of rollbacks) and R_{obs} (number of executed events, excluding coasting forward ones), the following expression is given for the time-optimal checkpoint interval of an LP

$$\chi = \left\lceil \sqrt{2 \frac{R_{obs}}{k_{obs}} \frac{\delta_s}{\delta_c}} \right\rceil \quad (1)$$

where δ_s and δ_c are, respectively, the average state saving time and the average time for an event in a coasting forward phase. Starting from expression 1, the authors derive an algorithm to calculate a new value of the checkpoint interval χ_n at the n-th observation period. Denoting with $\chi_{initial}$ the initial value for χ, with $\chi_{max} = 15$ an upper limit on χ, with χ_{min} the value resulting from equation 1 evaluated from the statistics collected during the last observation period, and with ρ a weighting parameter (generically assumed equal to 0.4), the authors propose the following adaptive solution:

if $n = 0$ **then** $\chi_n = \chi_{initial}$
else if $k_{obs} = 0$ **then** $\chi_n = \lceil (1-\rho)\chi_{n-1} + \rho\chi_{max} \rceil$
else $\chi_n = max(1, \lceil (1-\rho)\chi_{n-1} + \rho min(\chi_{min}, \chi_{max}) \rceil)$

Furthermore, an upper bound k_{max} on the number of rollbacks is introduced in order to enforce an LP experiencing frequent rollbacks to recalculate its checkpoint interval by using previous expression.

Fleischmann and Wilsey's Approach. The dynamic recalculation of χ proposed by Fleischmann and Wilsey [5] is based on the on-line observation of the following cost function:

$$E_c = C_{ss} + C_{cf} \quad (2)$$

where C_{ss} and C_{cf} represent, respectively, the CPU time spent by the LP in state saving and coasting forward operations. The adaptive checkpointing algorithm works as follows: at the first observation period χ is set to the initial value $\chi_{initial} = 1$; in the successive observation periods the cost function E_c is evaluated and χ is increased by one if E_c did not *significantly* increase. If the value of the cost function observed in the current period is greater than the one in the previous period, the adaptation direction is changed (χ is decreased by one). Furthermore, if during its whole history an LP never rolls back its checkpoint interval is set to χ_{max} (the authors select $\chi_{max} = 30$ as a suitable value in accordance with empirical observations).

In both previous solutions an upper limit on the value of χ is introduced due to the LP memory consumption problem. In fact, since rollback to GVT is possible, upon recovering 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 virtual 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 the frequent saturation of the input queue), thus making performance worse.

3 A Sparse State Saving Scheme

The state saving scheme we propose is based on the belief that the likelihood of a rollback point being contained in the virtual time interval I, the delimiting points of which are timestamps of successive events, increases with the distance between those points (a similar observation has been made in [21], with the difference that the starting point of the interval is the GVT of the simulation).

So, let us suppose that LVT^* is the current local virtual time of an LP (the current state of the LP is denoted as S_{LVT^*}), and the timestamp of the next event e_{next} to be scheduled is T^*. If $T^* - LVT^*$ is *significantly* larger than the usual difference between timestamps of successive already scheduled events, then, in all likelihood, either messages will arrive later having timestamps between LVT^* and T^*, or the sender of the message that schedules the event e_{next} at T^* will roll back undoing that message. So, in case of rollback, in all likelihood the rollback point will be contained in the interval $(LVT^*, T^*]$ and the state S_{LVT^*} will have to be restored due to rollback.

Based on this observation a checkpointing rule can be introduced to force the LP to record its state prior

[2]The GVT of a Time Warp simulation is defined as the minimum of the virtual times of all LPs and of the timestamps of messages in transit. Its notion is used to recover the storage allocated for obsolete information as no LP will ever rollback to a time before GVT.

to a local virtual time increment which generates an interval that is likely to contain a rollback point. If a rollback really occurs in that interval, no penalty due to coasting forward arises since the state to be restored is directly available.

Let $\bar{\Delta}_{LVT}$ be the average local virtual time increment due to the scheduling of a single event, and α be a real value grater than or equal to zero, the rule is synthesized by the following expression:

Upon scheduling e_{next} with timestamp T^*:
if $T^* - LVT^* > \alpha\bar{\Delta}_{LVT}$
then record current state;

The quantity $\alpha\bar{\Delta}_{LVT}$ is the threshold over which the local virtual time increment due to the scheduling of e_{next} determines an interval suspected to contain rollback points. Whenever the virtual time increment is less than this threshold, the current LP state is not recorded.

The choice of the value for α is far from being a trivial task because of its relation with both the distribution function of the local virtual time increment and the average rollback frequency of the LP. In what follows, an adaptive method for selecting suitable values for α is given.

3.1 Dynamic Selection of the Parameter α

Let us partition the execution of the LP into observation periods. Each period consists of L committed/rolled-back events (suggestions to choose L have been pointed out in [19]). Let α_n be the value of α at the n-th period, and $\bar{\Delta}^n_{LVT}$ be the average local virtual time increment (observed overall the scheduled events except the coasting forward ones) till the beginning of the n-th period. The value for $\bar{\Delta}^n_{LVT}$ is evaluated as follows:

$$\bar{\Delta}^n_{LVT} = \begin{cases} 0 & \text{if } n = 0 \\ \frac{1}{nL}\sum_{h=1}^{nL}\Delta^h_{LVT} & \text{if } n > 0 \end{cases} \quad (3)$$

where Δ^h_{LVT} is the virtual time increment due to the scheduling of the h-th event. During the n-th period the checkpointing activity of the LP is driven by the previous rule specialized as follows:

Upon scheduling e_{next} with timestamp T^*:
if $T^* - LVT^* > \alpha_n\bar{\Delta}^n_{LVT}$
then record current state;

The dynamic selection of α is based on the observation of the same cost function introduced by Fleischmann and Wilsey [5] (we report it below for convenience of the reader). At the n-th period, the cost function is evaluated as follows:

$$E_{c,n} = C_{ss,n} + C_{cf,n} \quad (4)$$

where $C_{ss,n}$ and $C_{cf,n}$ represent, respectively, the CPU time spent by the LP in state saving and coasting forward operations during that period.

The adaptive selection of α works as follows: at the first observation period α is set to zero, thus the state is recorded before processing each event. In the successive observation periods the cost function E_c is evaluated and α is increased by a quantity ϵ if E_c did not increase. Otherwise the adaptation direction of α is inverted (α is decreased by ϵ). The inversion of the adaptation direction takes place each time the last observed value of the cost function is greater than the previous one. So, the idea underlying the recalculation is to skip recording as much states as possible (by increasing the threshold over which the virtual time increment generates an interval likely to contain rollback points) until the prediction of the intervals containing rollback points becomes poor and the time spent due to coasting forward determines a performance decrease. In that case, the threshold is decreased until the checkpointing overhead becomes, in turn, the major factor adversely affecting performance.

Differently from [5], our approach tries to optimize the position of checkpoints in virtual time by exploiting the virtual time differences of the timestamps of arriving messages. This feature suggests that our solution, compared to the one in [5], allows the LP to skip recording more states before the cost function shows an increase due to coasting forward, with a benefic impact on the total checkpointing-rollback overhead. Experimental results reported in Section 4 confirm this hypothesis.

When the value of α increases, the number of states recorded as checkpoints usually decreases, thus also C_{ss} decreases. If the LP never rolls back (or it shows a very low rate of rollbacks) the inversion of the adaptation direction of the parameter α may never take place (or it takes place only when very few states are recorded in each period). As outlined in Section 2, this behavior is not desirable due to the LP memory consumption problem. Therefore, in the case of very low rollback frequency, the inversion of the adaptation direction has to be imposed in order to have at least a percentage γ of states recorded in each period. To this purpose, each LP keeps track of the number of states NC_n recorded as checkpoints during the n-th period; at the end of the period it compares such a number with the minimum number γL of states to be recorded. If $NC_n < \gamma L$ the adaptation direction is inverted (α is decreased by ϵ). We adopt the value $\gamma = 1/15$, thus, on average, the LP is not allowed to skip recording its state plus than 14 times over 15.

The value of ϵ determines how greatly the cost function E_c changes during the execution. Preliminary results showed that small values allow to avoid too large oscillations of E_c and usually generate a better performance compared to large values (experiments presented in Section 4 have been realized by adopting the

value 0.05). Below the complete structure of the algorithm for the selection of α at the n-th period is shown (the boolean flag fwd indicates whether the adaptation direction actually goes towards increasing values of α or not):

if $n = 0$ **then** $\alpha_n = 0$;
if $n = 1$ **then** $\{\alpha_n = \alpha_{n-1} + \epsilon;\ fwd = TRUE\}$;
if $n > 1$ **then**
  if fwd **then**
    if $(E_{c,n-2} < E_{c,n-1})$ or $(NC_{n-1} < \gamma L)$
    then $\{\alpha_n = max(0, \alpha_{n-1} - \epsilon);\ fwd = FALSE\}$
    else $\alpha_n = \alpha_{n-1} + \epsilon$
  else
    if $(E_{c,n-2} < E_{c,n-1})$ or $(\alpha_{n-1} = 0)$
    then $\{\alpha_n = \alpha_{n-1} + \epsilon;\ fwd = TRUE\}$
    else $\alpha_n = max(0, \alpha_{n-1} - \epsilon)$

4 A Performance Comparison

In this section experimental results are presented to compare performance achievable by using the checkpointing scheme proposed in this paper (hereafter Q) to the one obtained by using Ronngren and Ayani's approach (RA) and Fleischmann and Wilsey's approach (FW).

The experiments have been carried out by using the distributed simulation platform SIMCOR [2], which has the following basic characteristics: LPs are statically assigned to processors; the aggressive approach [10] is adopted in the cancellation phase (i.e., antimessages are sent as soon as an LP rolls back); a single scheduler (Time Warp kernel) runs on each processor and manages the local event list by scheduling local LPs according to the STF (Smallest-Timestamp-First) algorithm; LPs execute the coasting forward in atomic fashion.

We propose simulation results of two synthetic workloads. The first, known as PHOLD, has been widely used for testing performance of Time Warp simulators (it was originally described in [8]). In this workload, a constant number of messages circulate among the LPs, messages are equally likely to be forwarded to any other LP and the timestamp increments are taken from some stochastic distribution. We consider a PHOLD model with 32 LPs and 1 message for each LP at the simulation starting. Timestamp increments are taken from an exponential distribution with mean 1. The second workload (HOT SPOT) is a modification of the first. It has 4 hot spot LPs to which 50% of all messages are routed. The experiments were performed on 4 nodes (386i 16MHz) of an iPSC/2 multiprocessor. The experiments have been carried out by adopting four different values (0.1, 0.25, 0.5 and 1.0) of the ratio δ_s/δ_c between the average state saving time and the average event routine time (the state saving cost is fixed at 300 $\mu sec.$ and the value of the ratio is modified, as in [16], by introducing variable delay loops into the event routine). In this way we can compare Q, RA and FW either when the average event routine cost dominates or when it is comparable to the average state saving time. The observation period was fixed at 500 events and, as mentioned in Section 3, the value 0.05 was chosen for ϵ.

We report measures related to: (a) the number of checkpoints NC taken by the LPs, (b) the average coasting forward length ACF, (c) the event rate (i.e., committed events per second), (d) the efficiency (i.e., the ratio between the total number of committed events and the total number of processed events, excluding coasting forward ones), (e) the total execution time ET and (f) the average distance between checkpoints (i.e., the average checkpoint interval ACI). We base our comparison on the average value analysis. All parameter values result as the average of 10 runs. At least 10^6 committed events were simulated in each run.

4.1 Results of the Experiments

Five series of plots have been reported. The first series (Figure 1) shows the ratio between the number of checkpoints taken by RA and FW and the number of checkpoints taken by Q ($NC(Q)$). The second series (Figure 2) shows the ratio between the average coasting forward of RA and FW and the average coasting forward of Q ($ACF(Q)$). The third and the fourth series (Figure 3 and Figure 4) report, respectively, the efficiency and the event rate for all the checkpointing schemes. The fifth series (Figure 5) shows the ratio between the execution times of RA and FW and the execution time of Q ($ET(Q)$). The sixth series (Figure 6) reports the average distance between checkpoints for all the checkpointing schemes. In each series results are reported while varying the ratio between the average state saving time and the average event routine time (the x axis has a logarithmic scale).

PHOLD Workload. The checkpointing overhead of RA is between 0.72 and 0.93 times that of Q. The minimum value of $NC(RA)/NC(Q)$ is obtained for small state saving time compared to the event routine time (i.e., $\delta_s/\delta_c = 0.1$). In this point, due to expression 1, the LPs tend to choose small checkpoint intervals under RA (the measured value of the average checkpoint interval under RA is around 2.21 events); hence, the larger checkpointing overhead of Q means very small average distance between checkpoints (the measured value is around 1.86 events) that, together with the attempt to optimize their position in virtual time with respect to the rollback points, lead to very small ACF under Q. As a result, in this point the average coasting forward of RA reaches its maximum distance from $ACF(Q)$ ($ACF(RA)$ becomes around 11 times $ACF(Q)$).

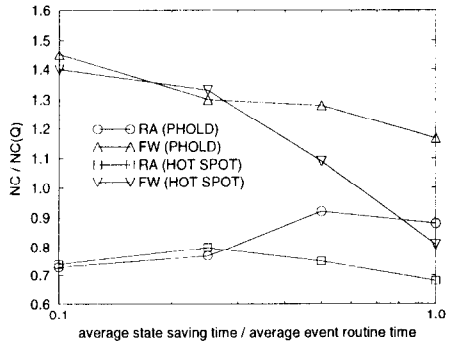

Figure 1. Ratio Between the Number of Checkpoints of RA and FW and the Number of Checkpoints of Q.

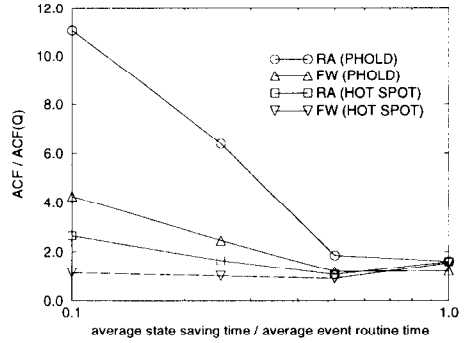

Figure 2. Ratio Between the Average Coasting Forward of RA and FW and the Average Coasting Forward of Q.

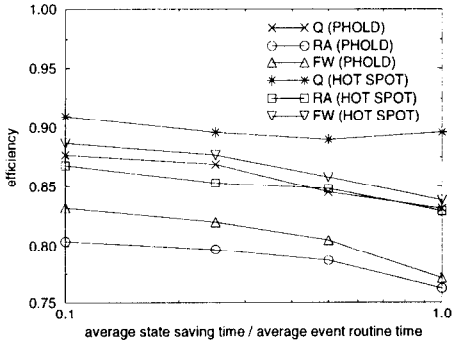

Figure 3. Efficiency.

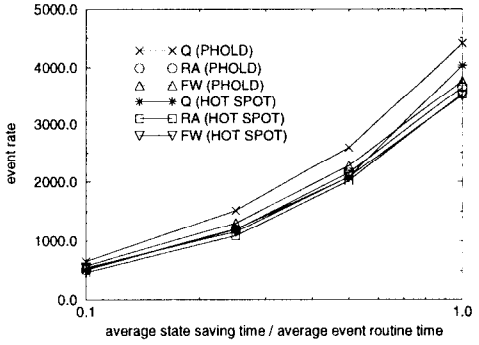

Figure 4. Event Rate.

The difference between the checkpointing overhead of RA and that of Q decreases when $0.5 \leq \delta_s/\delta_c \leq 1.0$ (the minimum distance, that is 7%, is obtained when $\delta_s/\delta_c = 0.5$) and $ACF(RA)$ becomes between 1.9 and 1.6 times $ACF(Q)$. The efficiency of Q is between 7% and 10% greater than the efficiency of RA. This behavior directly derives from the longer ACF of RA compared to Q. In fact, as shown in [16], the drawback usually incurred due to long average coasting forward length is an increase in the number of rollbacks (thrashing) which adversely affects the efficiency of the simulation. The efficiency of Q gets its maximum gain when state saving and event routine have the same cost. In this point, the measured value of the average checkpoint interval under RA is around 5 events and $ACF(RA)$ is 2.07 events while $ACF(Q)$ is 1.18 events with an average checkpoint interval of 4.36 events. In conclusion, the observed reduction of ACF allows Q to mantain a good efficiency in the whole considered range for δ_s/δ_c (i.e., the execution under Q is affected by thrashing less than under RA). The plots related to the event rate and to the execution time match with previous results. In particular, the event rate of Q is between 20% and 26% greater than the event rate of RA and the execution time of RA is between 20% and 26% larger than $ET(Q)$.

The behavior of FW is quite different from that of RA. The major difference is a larger checkpointing overhead with the result that the number of checkpoints of FW is between 1.16 and 1.45 times the number of checkpoints of Q. In spite of this, $ACF(FW)$ remains larger than $ACF(Q)$ in the whole range for δ_s/δ_c, with a maximum distance when $\delta_s/\delta_c = 0.1$ (in this point the measured value of the average checkpoint interval of FW is around 1.49 events while, as mentioned before, $ACI(Q)$ is 1.86). The efficiency of FW is slightly greater than the efficiency of RA, especially for large event routine time compared to the state saving time. This behavior derives because FW shows a notably smaller ACF compared to RA (with a maximum distance when $\delta_s/\delta_c = 0.1$) denoting less thrashing. However the efficiency of FW remains smaller than that of Q. This phenomenon, together with the larger checkpointing overhead and coasting forward length,

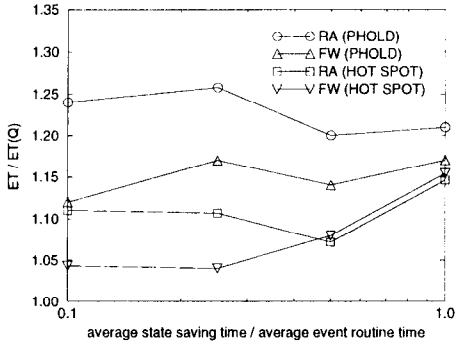

Figure 5. Ratio Between the Execution Time of RA and FW and the Execution Time of Q.

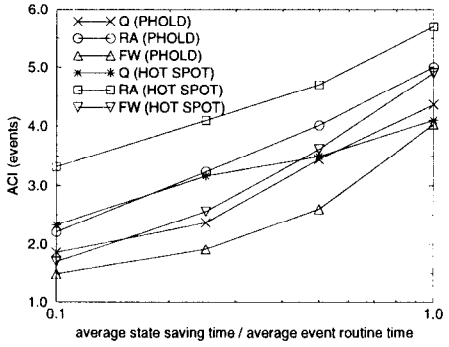

Figure 6. Average Checkpoint Interval (ACI).

lead the event rate of FW to be smaller (between 12% and 17%) than that of Q, hence the execution time of FW is between 12% and 17% larger than $ET(Q)$.

HOT SPOT Workload. In the range $0.1 \leq \delta_s/\delta_c \leq 0.25$ the difference between the checkpointing overhead of RA and that of Q is similar to the one observed in the case of the PHOLD workload. On the contrary, in the range $0.5 \leq \delta_s/\delta_c \leq 1.0$ the results are quite different ($NC(RA)/NC(Q)$ remains around 0.7). As for the PHOLD workload, $ACF(RA)$ is larger than $ACF(Q)$ but the distance is not so evident; in particular, $ACF(RA)$ is between 1.3 and 2.7 times $ACF(Q)$. The difference between the efficiency of RA and that of Q is reduced compared to the one observed for the PHOLD workload, however, Q remains between 4.5% and 8% more efficient than RA. As for the PHOLD workload, the maximum distance between the efficiency of RA and that of Q is noted when $\delta_s/\delta_c = 1.0$. In conclusion, when $\delta_s/\delta_c = 1.0$ the event rate of Q is up to 14% greater than the event rate of RA, hence $ET(RA)$ is up to 14% larger than $ET(Q)$. On the contrary, for smaller values of δ_s/δ_c the performance difference between the two schemes looks slightly reduced; the event rate of Q is around 11% greater than the event rate of RA, so $ET(RA)$ is around 11% larger than $ET(Q)$.

While moving from $\delta_s/\delta_c = 0.1$ to $\delta_s/\delta_c = 1.0$ the checkpointing overhead of FW decreases from 1.4 to 0.82 times the checkpointing overhead of Q and $ACF(FW)$ is between 1.1 and 1.6 times $ACF(Q)$. This means that Q achieves around the same cost of coasting forward (or even smaller) but usually with less checkpoints (i.e., with larger average checkpoint intervals) compared to FW (this feature derives directly from the better placing of checkpoints of Q). The efficiency of FW is between 2% and 7% smaller than the efficiency of Q (the maximum distance is noted when $\delta_s/\delta_c = 1.0$). So, the basic factors affecting performance are the larger checkpointing overhead and the smaller efficiency of FW. The event rate of Q is between 4% and 15% greater than the event rate of FW (the maximum distance is noted when $\delta_s/\delta_c = 1.0$), so $ET(RA)$ is between 4% and 15% larger than $ET(Q)$.

5 Conclusions

In this paper a sparse state saving scheme for Time Warp simulators is presented. The scheme aims at establishing a relation between checkpoints and rollback points in order to reduce the average rollback cost. To this purpose, statistics on the event history are collected for predicting states that will be restored due to rollback. Those states are recorded as checkpoints in order to make them directly available. The percentage of states to be recorded is a function of a parameter α whose value determines the degree of goodness of the prediction. The adaptation of α is realized by observing the impact of its value on a checkpointing-rollback cost function. Our state saving scheme attempts for the first time to put together the exploitation of statistics related to the virtual time progression with the observation of the CPU time spent due to checkpointing and rollback operations, in order to determine both the best appropriate percentage of states to be recorded as checkpoints and also their position in virtual time.

Two synthetic workloads, PHOLD and HOT SPOT, have been used to provide a performance comparison with previous solutions. In order to carry out the comparison while varying the impact of checkpointing and coasting forward on the execution time, simulations have been realized by using several different values of the ratio between the average state saving time and the average event routine time of the logical processes.

For both simulation models, the new scheme shows an improvement of the efficiency and of the event rate compared to other schemes. Hence the results are decidedly encouraging thus pushing towards future work focused on both a more accurate performance comparison, based on a broad variety of simulation workloads (including real world simulation models), and also an

investigation onto the impact of the new approach on memory space.

As a final point, we would like to outline one limitation of our state saving scheme. It exploits information only about the arithmetic mean of the average virtual time increment of an LP. This approach has the advantage of adding negligible overhead to the execution but cannot well tackle situations in which the virtual time increment follows a skewed, non-unimodal distribution. In those situations the scheme might not produce performance improvement over previous proposals. In order to overcome this problem, sophisticated forecast methods would be adopted. The embedding of some of those methods in our state saving scheme is the object of a study in progress.

Acknowledgments. The author would like to thank Cristina Auriche and the anonymous referees for their helpful comments and suggestions.

References

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

[2] B. Ciciani and M. Angelaccio, "An Interface to Develop Time-Warp Based Parallel Simulations", *Proc. Massively Parallel Processing Conference*, pp.20-21, June 1994.

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

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

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

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

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

[8] R.M. Fujimoto, "Performance of Time Warp Under Synthetic Workloads", *Proc. Multiconf. Distributed Simulation*, Vol.22, No.1, January 1990.

[9] 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.

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

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

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

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

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

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

[16] 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.

[17] F. Quaglia and L.R.G. Auriche, "A New Technique for Adaptive Checkpointing in Time Warp", *Proc. 11-th European Simulation Multiconference*, pp.35-39, June 1997.

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

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

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

[21] J. Steinman, "Breathing Time Warp", *Proc. 7-th Workshop on Parallel and Distributed Simulation*, pp.109-118, May 1993.

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

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