ELSEVIER Simulation Practice and Theory 6 (1998) 461–478

SIMULATION PRACTICE AND THEORY

Run-time selection of the checkpoint interval in Time Warp based simulations

Laurent R.G. Auriche, Francesco Quaglia [1], Bruno Ciciani *

Systems and Computer Engineering Department, University of Roma "La Sapienza", Via Salaria 113, 00198 Rome, Italy

Received 21 January 1997; revised 25 July 1997

Abstract

Time warp discrete event simulators take advantage of the parallel processing of simulation events. On the other hand, they suffer from the overhead required to enforce the causality relation. This overhead consists of the time for saving the states of logical processes, the time for the rollback procedures and the wasted simulation time, that is the time spent for the processing of events which are undone because of rollback. Two techniques have been developed for state saving: periodic and incremental. In this paper we study the periodic technique, and we present an analytical model describing the simulation execution time in function of both the state saving cost and the rollback cost. Furthermore, we derive a methodology that allows each logical process to adapt its state saving period on line in order to reduce the simulation execution time. Experimental results show that, in some simulation scenarios, our methodology improves performance, in comparison to already existing proposals. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Time Warp modeling; Adaptive checkpointing; Parallel simulation application; Performance optimization

1. Introduction

A discrete event simulation consists of a sequence of events occurring in virtual time. The execution of each event eventually produces new events to be processed later. Strategies for controlling the correct order of execution are easily realizable by using centralized systems with a single event queue. Nevertheless, since simulation can be time consuming, it seems natural to attempt to use multiple processors for speeding up the simulation process.

In parallel discrete event simulation, the parts constituting the system to be

* Corresponding author. E-mail: ciciani@dis.uniroma1.it.
[1]E-mail: quaglia@dis.uniroma1.it

simulated are modeled by logical processes (LPs) which communicate by exchanging timestamped messages [6]. The scheduling of one event for an LP at time t is simulated by sending a message with timestamp t to the LP.

Each LP has its own simulation clock (*local virtual time*), and an event list (*input queue*) in which incoming messages are enqueued, sorted by increasing timestamp values. Upon processing a message, an LP sets its own simulation clock to the timestamp of that message. Simulation results are correct if each LP processes received messages in non-decreasing timestamp order (*causality constraint*).

The distributed scheduling of events may affect correctness of the simulation unless a synchronization protocol is introduced. The main synchronization protocols to satisfy the causality constraint are the Chandy–Misra protocol [3] and the Time Warp protocol [8,9]. The former is blocking (each LP suspends its execution until it knows that no message will arrive with a timestamp smaller than the one of the message that is going to be scheduled) thus limiting the exploitation of parallelism. The latter, being non-blocking, is well suited for simulation of systems with high degree of intrinsic parallelism.

Under Time Warp each LP executes asynchronously. It extracts messages from its input queue and processes them without undergoing any constraint, by optimistically assuming that the event scheduling is correct. As a consequence, local simulation clocks of the LPs may diverge, thus an LP may receive a message with a timestamp in the past (*straggler* message). If this occurs, the causality constraint is broken, and a procedure for recovering from this violation is executed. Such procedure consists in undoing the over-optimistic portion of the simulation by rolling back the LP to a previous state. From this state the LP resumes its computation.

The implementation of the rollback mechanism requires the ability of restoring, at run time, a past state of an LP. To this purpose two main state saving mechanisms have been proposed: *periodic* [2,11] and *incremental* [1,16]. According to the first solution, a subset of LP states are saved (*checkpoints*) into a *state queue*, whereas, in the second, the inverse of all the incremental changes of an LP state are recorded.

Both solutions, in their turn, affect the rollback cost. In the first case the LP can be pushed to a state further back than is really necessary for the elaboration of the straggler message, and a re-execution of some intermediate events (*coasting forward*) is required to correct the difference. The coasting forward phase can be executed either in atomic fashion (*batch* execution) or not. In the case of incremental state saving, the state restoration is accomplished by applying, one at a time, the saved incremental changes in backward order, and its cost thus strongly depends on both the rollback length and the time for a single back step of the state reconstruction. It has been proved in Ref. [12] that, usually, incremental state saving improves performance over periodic state saving only when both the fraction of the state modified due to the execution of one event and the rollback length are minimal.

In the case of periodic state saving, analytical models have been proposed [5,11,12,15] to describe the simulation execution time as a function of the checkpoint interval (i.e., the number of events processed between successive checkpoints). However, all of them neglect the *invalid-state* cancellation cost (an invalid-state is a checkpointed state that must be removed from the state queue because an older one has been restored due to rollback). This cost depends on two factors: the state queue implementation and the system memory management. Dealing with the first factor, if dynamic implementation is adopted, invalid-state cancellation requires explicit memory release which, in its turn, needs either calls to library routines or, in the worst case, context switches to the operating system. In both cases, the outcoming time consumption would not be negligible. Dealing with the second factor, virtual memory management introduces I/O operations that can affect the invalid-state cancellation cost when a high page fault rate occurs. This phenomenon can be present in large simulation applications or in simulation experiments running on a cluster of workstations which support other user applications.

In this paper we study the periodic state saving technique when batch coasting forward is adopted, and introduce an analytical model describing the LP execution time versus the checkpoint interval. The model takes into account the invalid-states cancellation cost. Furthermore, our study leads to an equation that defines the feasible domains for the LP rollback behavior parameters and that, once fixed a forecast number of committed events, states the relation between the total number of executed events (committed plus rolled back) and the values of such parameters.

As for previous models, our analysis also shows the checkpoint interval which minimizes the execution time depends on the LP rollback behavior parameters (i.e., the rollback probability and the rollback length). Usually, these parameters cannot be analytically evaluated, thus requiring their on-line measurement.

In the class of simulation experiments in which the rollback behavior parameters reach steady values, and both the *throttling* and the *thrashing* mechanisms [14] can be neglected, a single measure suffices for the selection of the optimal checkpoint interval (see Section 2 for details about throttling and thrashing). This is because the changing of the checkpoint interval to the value that is optimal in relation to the measured rollback parameters, does not generate the collateral effect of modifying their values. However, the absence of a criterion for establishing a priori if an experiment belongs to such a class, pushed some authors to introduce adaptive algorithms to dynamically adjust the checkpoint interval on-line in order to achieve better optimization results compared to static periodic state saving [13,15].

We also derive a methodology which allows each LP to adapt its checkpoint interval to its own rollback behavior on-line. The innovation of our methodology is that it tries to predict the LP rollback behavior parameters on-line; afterward, it uses their forecast values for the selection of the checkpoint interval. The prediction is carried out by using information about the past LP behavior and an analytical extrapolation technique.

Our approach is carefully compared to the one in Ref. [15] and experimental results show that our adaptive methodology improves performance in some simulation scenarios. A discussion clarifies when it is profitable to adopt our methodology rather than that described in Ref. [15].

The paper is organized as follows: Section 2 describes Time Warp basic concepts when periodic state saving is adopted; in Section 3 the model and the adaptive algorithm presented in Ref. [15] are investigated; Section 4 describes both our model and our adaptive checkpointing algorithm; experimental results in Section 5 validate our analysis; a short conclusion constitutes Section 6.

2. Time Warp basic concepts

As already outlined in Section 1, LPs interact exclusively by message exchange. Each message is stamped with a virtual send time and a virtual receive time. Virtual send time is the local virtual time of the sender when the message is sent. Virtual receive time, named timestamp, is the virtual time at which the receiver has to process the message.

The run-time support of an LP has the following components [10]:

– *process name*, that identifies the LP;

– *local virtual time* (LVT), that is the timestamp of the last message processed by the LP;

– *current state*, that groups the values of variables representing simulation state at the current LVT;

– *state-queue*, that stores copies of some LP states;

– *input-queue*, that stores incoming messages, sorted according to increasing values of their timestamps (some of these messages have already been processed, but they are not discarded because they could be processed again if a rollback occurs);

– *output-queue*, that stores messages that the LP has sent, ordered according to increasing values of virtual send time.

The resulting structure of an LP is shown in Fig. 1. When an LP processes a message, it moves its LVT to the timestamp of that message, updates some of its state variables and eventually sends output messages. If a message is received with a timestamp smaller than the current LVT (causality violation), the LP rolls back to its state immediately prior the causality violation, and resumes execution from that state.

In the following subsections, we give details about scheduling policies among LPs that share the same processor, rollback and state restoration mechanisms and pro-

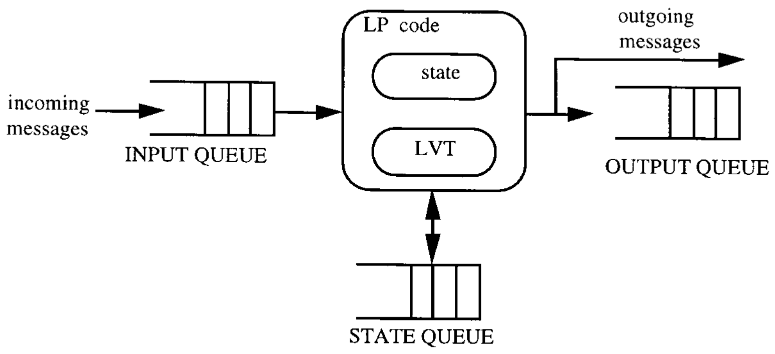


Fig. 1. Structure of an LP.

cedures for recovering storage allocated to obsolete recorded states and buffered messages.

2.1. Scheduling policies

Usually, in parallel discrete event simulation, more than one LP runs on each processing element. The scheduling of the LPs that share the same processor can be actuated either by the underlying operating system or by a Time Warp scheduler loaded on each processor. Three basic Time Warp schedulers have been defined [8].

The first, known as *Smallest-Timestamp-First* (STF) scheduler, gives higher priority to the LPs having messages to be processed with lower timestamps. In particular, the STF scheduler chooses for execution the LP having the message to be processed with the smallest timestamp.

The second, referred to as *Smallest-Virtual-Time* (SVT) scheduler, chooses for execution, from among the LPs having messages to be processed, the LP having the smallest LVT, which is allowed to process one message.

The third, named *Round-Robin* (RR) scheduler, chooses one LP for executing without basing on the LVTs or on the timestamp of messages to be processed. In particular, LPs that are ready to execute (i.e., that have at least one message to be processed) share the CPU in a round-robin fashion. During its turn, an LP processes one message.

2.2. Cancellation strategies

Upon rolling back, the LP must cancel the effects of some sent output messages. This is realized by sending an antimessage for each output message which must be undone (*cancellation phase*). When an antimessage is received, the receiving LP searches for the corresponding message in its input queue and, upon finding it, both of them are discarded. If the original message has already been processed the receiving LP must roll back.

Two approaches have been proposed to implement the cancellation phase: *aggressive* and *lazy*. When the aggressive approach is used, antimessages are sent immediately when a rollback occurs. When the lazy approach is used, antimessages are not sent when a rollback occurs, but they are placed into a queue of pending antimessages. When the LP resumes its execution, it will generate again output messages. If one of these messages is the same as one that would have been canceled during rollback, then this message and the corresponding antimessage are discarded. A complete description of cancellation strategies can be found in Ref. [7]. In Ref. [14], an extensive simulation study has been proposed to point out the relation between cancellation strategies and the simulation execution time.

2.3. State restoration

The rollback of an LP requires the ability of regenerating the state of the LP immediately prior to the causality violation. To this purpose, a simple mechanism, known as *copy* state saving has been proposed [8].

When copy state saving is adopted, the state of the LP is saved into the state queue before a new event is executed. In this way, all the states passed through by an LP are available, and the rolling back of the LP to a virtual time *T* is realized by simply restoring the most recent state with LVT smaller than *T*. Upon rolling back, the LP cancels from its state queue all the states with LVT larger than or equal to *T* (cancellation of invalid-states). This technique adds to each event execution a checkpointing overhead, which is quantified by the sum of the time required to allocate a state buffer and the time required to copy the current LP state into the buffer.

A promising approach for reducing such overhead is to perform state saving each x event executions (x being the checkpoint interval of an LP) [2]. This solution, known as *periodic* state saving, has both beneficial and detrimental effects on the LP execution time. It reduces the number of CPU cycles spent in state saving operations, but upon rolling back, the required state may not be into the state queue. In the latter case, such state must be recomputed by starting from a previous saved one and by reprocessing input messages (thus adding a time penalty). An LP that is recomputing a missing state is said to be in a *coasting forward* phase [6]. This phase simulates the LP forward again from the restoration point to the virtual time when it can correctly receive the rollback generator message (straggler or antimessage). Coasting forward events are different from normal events, because they are executed only to modify the process state, without the re-generation of output messages (these messages have been sent previously). Two approaches are possible to implement this phase. In the first one (batch execution [6,15]), coasting forward events are executed altogether in atomic fashion during the rollback, so they cannot generate checkpoint insertion. In the second one (non-batch execution [11]) coasting forward events are scheduled as normal events (except for the generation of output messages) so they can generate checkpoint insertion.

These approaches lead to different relations between events and checkpoints. In fact: in the first case the number of scheduled events between two successive checkpoints is always equal to x [15]. Meanwhile, in the second case, this number can vary from 1 and x [11]. This difference is shown in Fig. 2.

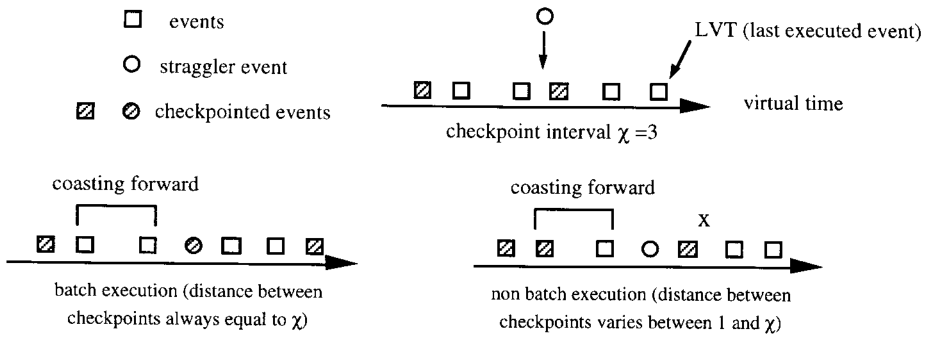

events
straggler event
checkpointed events
LVT (last executed event)
virtual time
checkpoint interval χ =3
coasting forward
batch execution (distance between checkpoints always equal to χ)
coasting forward
X
non batch execution (distance between checkpoints varies between 1 and χ)

Fig. 2. Relation between events and checkpoints for different implementations of the coasting forward phase.

If a Time Warp scheduler runs on each processor and batch coasting forward is adopted, the processor is prevented from executing events in the other LPs until a rollback operation is finished. When the checkpoint interval increases, the average number of intermediate states which must be crossed in a coasting forward phase, before the required state is obtained, also increases. As a consequence the average execution time of each rollback increases. This phenomenon generates both advantages and drawbacks.

When the rollback cost increases more messages and antimessages, destined to LPs that share the same processor of the rolling back LP, arrive before any of the LPs act. As a result, the increase of the checkpoint interval tends to defer action until a more correct action can be taken and, thus, the number of rollbacks decreases. In Ref. [14], Preiss et al. defined this mechanism as *throttling*.

On the other hand, when the rollback cost increases, the differences between the LVTs of local LPs and the LVTs of remote ones increase (assuming that the remote processor is not simultaneously rolling back). As these differences increase, there is an increasing probability that some of the remote LPs will receive a message (or antimessage) after the rollback operation has been completed, and this can lead to a significant increase in the number of rollbacks. This phenomenon is the *thrashing* mechanism defined in Ref. [14].

Experimental results presented in Ref. [14] show that when the rollback cost increases slightly, the throttling mechanism appears; on the other hand, when the rollback cost significantly increases (i.e., for large values of the checkpoint interval) the thrashing mechanism dominates.

These phenomena show a feedback effect between the changing of the checkpoint interval and the rollback behavior which necessarily push towards adaptive checkpointing algorithms in order to achieve performance optimization.

2.4. Global virtual time and fossil collection

The *Global Virtual Time* (GVT) of a Time Warp simulation, is the minimum value among the LVTs of the LPs and the timestamps of messages in transit (i.e., not yet processed). Thus, GVT indicates the progress of the simulation, because no LP can roll back to a virtual time that is less than GVT. If periodic state saving is adopted, for each LP the saved state with the largest LVT less than or equal to GVT must be preserved for executing an eventual rollback procedure, and with it all the messages with timestamps larger than its LVT. States and messages that can be discarded are called fossils, and the procedure of recovering storage allocated to fossils is called *fossil collection*. Such procedure introduces a time penalty which is the sum of the time to estimate the GVT value and the time to correctly release buffers allocated to fossils.

3. Related work

Because our paper deals with the problem of periodic state saving when batch coasting forward is adopted, in this section we investigate the study proposed in Ref. [15], which analyzes exactly the same problem.

In their paper, the authors propose a model describing the LP execution time in function of the checkpoint interval. The model is based on the following assumptions:

(1) non-preemptive rollback is considered [6];
(2) there is no correlation between when in simulated time rollbacks occur and the timestamps of events that are checkpointed;
(3) the batch mode is used in the coasting forward;
(4) the execution time of fossil collection is negligible;
(5) the invalid-states cancellation cost is negligible;
(6) the changes in the checkpoint interval only marginally affect the rollback behavior.

Assumption (1) allows the authors to consider an event execution as an atomic action (message/antimessage preemption is discarded). Given assumptions (2) and (3), the average length of the coasting forward phase is assumed equal to $(x-1)/2$; this is because, given assumption (2), the probability distribution function of the length of this phase is assumed uniform and, given assumption (3), the distribution is between 0 and $x-1$.

Assumptions (4) and (5) are not explicitly declared but they can be derived from expression (2) in Ref. [15]; in fact, in this expression, which describes the simulation execution time, no term takes the fossil collection time and the invalid-states cancellation cost into account (we recall that invalid-states could be removed at each rollback, and in some state queue implementations the cancellation cost could not be negligible).

Assumption (6) is introduced to derive an adaptive algorithm for the on-line adjustment of the LP checkpoint interval.

In Ref. [15], once measured, the parameters k_{obs} (number of rollbacks), R_{obs} (number of executed events), and once fixed d_s (average state saving time) and d_c (average execution time for an event in a coasting forward phase), the following expression for the optimal checkpoint interval is given:

$$x = \left\lceil \sqrt{2\,\frac{R_{obs} d_s}{k_{obs} d_c}} \right\rceil \tag{1}$$

Starting from Eq. (1), the authors derive an algorithm to calculate a new estimate checkpoint interval x_n at the n-th observation period, for each LP. Denoting with $x_{initial}$ the initial value for x, with x_{max} an upper limit on x, with x_{min} the value resulting from Eq. (1) evaluated from the statistics gathered in the last observation period, and with r an appropriate weighting parameter (generically assumed equal to 0.4), the authors propose the following adaptive algorithm:

$$\begin{aligned} x_n = &\ \textbf{if } n == 0 \textbf{ then } x_{initial} \\ &\ \textbf{else if } k_{obs} == 0 \textbf{ then } \lceil (1-r)\cdot x_{n-1} + r\cdot x_{max} \rceil \\ &\ \textbf{else } \max(1, \lceil (1-r)x_{n-1} + r \min(x_{min}(k_{obs}, R_{obs}, d_s, d_c), x_{max}) \rceil) \end{aligned} \tag{2}$$

An upper bound on the number of rollbacks, k_{max}, is introduced to enforce an LP experiencing frequent rollbacks to recalculate its checkpoint interval using Eq. (2). The value of x_{max} is assumed equal to 15. This choice is made because of the LP memory consumption problem. In fact, as shown in Refs. [14,15], if the checkpoint interval is increased, less memory is required to store checkpointed states but, when fossil collection is performed, more memory is needed to retain messages with a timestamp between the GVT and the LVT of the last checkpointed state with virtual time less than GVT. This phenomenon could increase the fossil collection frequency, thus making performance worse.

The rationale behind the adaptive algorithm in Ref. [15] is to carefully use information about the past behavior of a process (i.e., the old value of the checkpoint interval and the ratio R_{obs}/k_{obs} measured at the end of the last observation period) to select the new checkpoint interval. This choice seems to be very good when the changes in the rollback behavior do not require the selection of a new checkpoint interval which is very different from the previous one. As an alternative approach, the methodology we propose in this paper uses information about the past behavior of an LP only for the prediction of the rollback parameters of the following observation period. Afterward, only these forecast values are considered for the selection of the new checkpoint interval (disregarding errors eventually produced in the prediction phase).

We will show that neither our adaptive methodology nor that in Ref. [15] can be considered optimal in any case. The choice of one or the other approach depends on the particular simulation experiment. In Section 5 we will define in which simulations our methodology is more convenient than the one in Ref. [15] and vice versa.

4. Adaptive checkpointing

In this section we build an analytical model of the LP execution time in a Time Warp simulation and then we derive an adaptive checkpointing algorithm for the optimization of the execution time versus the checkpoint interval of the LP.

4.1. The model

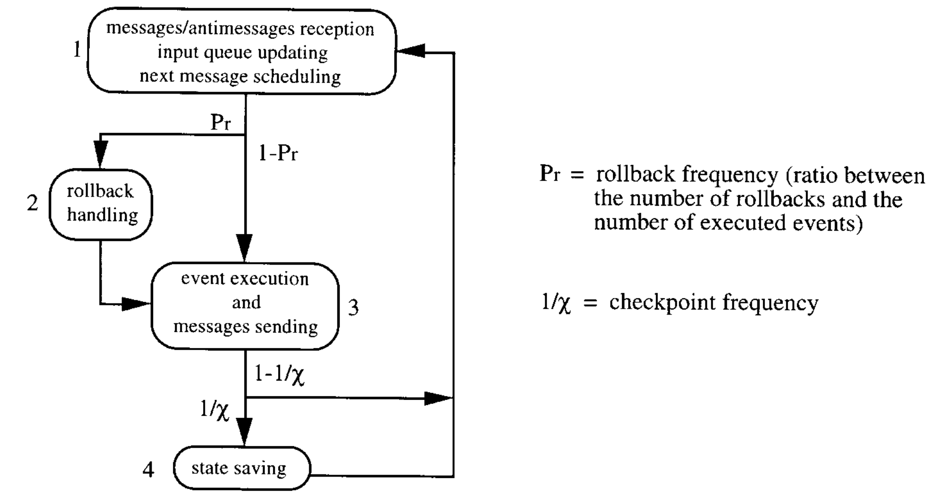
We adopt the same assumptions from (1) to (4) as in Ref. [15]. Furthermore, we introduce the following assumption:

(5) aggressive approach is adopted in the cancellation phase.

In Fig. 3 we describe the characteristic execution loop of a generic LP under assumptions (1), (3), (4) and (5). The loop depicts the logical activities to be performed whether a scheduler, responsible for the queues updating and the scheduling phase, is present or not. If the scheduler is not present, each LP is responsible for its queues updating and for the extraction of its next message to be processed. Without losing generality, in our analysis we award all the logical activities to the LP.

The LP activities can be grouped into four blocks:

– in block 1 the LP checks the presence of messages/antimessages addressed to itself; if any, it enqueues them updating its input queue and schedules the next



1
messages/antimessages reception
input queue updating
next message scheduling
Pr
1-Pr
2
rollback handling
event execution and messages sending
3
1-1/χ
1/χ
4
state saving
Pr = rollback frequency (ratio between the number of rollbacks and the number of executed events)
1/χ = checkpoint frequency

Fig. 3. Characteristic execution loop.

message (these operations are executed in an established point of the execution loop because we are considering non-preemptive rollback [6], so the arrival of new messages/antimessages can never interrupt the current event execution);

– in block 2 the LP handles the rollback procedure eventually related to the currently scheduled message;

– in block 3 the LP executes the event routine for the currently scheduled message and sends (and enqueues in its output queue) the output messages eventually produced during this execution;

– in block 4 the LP records its current state into the state queue (checkpoint insertion).

A single execution loop corresponds to the execution of a single event. Denoting with N_{tot} the total number of LP simulation events, with T_{tot} the execution time required by the LP during the whole simulation, and with T_e the average time for the execution of a single loop (i.e. of a single event), we can write:

$$T_{tot} = N_{tot} T_e \tag{3}$$

We remark that the quantity N_{tot} includes committed and rolled back events, but does not include the coasting forward ones; in fact, given Eq. (3), these events are executed in block 2 and so they do not affect the number of execution loops. Denoting with T_i the average time spent into the i-th block of the execution loop, we can write:

$$T_e = T_1 + P_r T_2 + T_3 + \frac{1}{x} T_4 \tag{4}$$

Let us explain the structure of the terms in Eq. (4). We denote with N_{in}, the average number of messages received in each execution loop; N_{ant}, the average number of antimessages received in each execution loop; d_{in}, the average time required to insert a message (or antimessage) in the input queue; d_{sc}, the average time for the scheduling of the next message; d_{out}, the average message (or antimessage) sending time; d_d, the average time required for the cancellation of an invalid-state; d_r, the average time to restore a valid state; d_c, the average event routine computation time; d_s, the average state saving time; f_0, the average number of output messages produced as a result of the elaboration of a single message; L_r, the rollback length, i.e. the average number of event executions undone by a single rollback occurrence.

The expression for T_1 becomes:

$$T_1 = (N_{in} + N_{ant})d_{in} + d_{sc} \tag{5}$$

The expression for T_2 is the sum of the following terms:

– the first term, which takes care of the antimessages sending time, is equal to: $L_r f_0 d_{out}$; in fact, having an average number of L_r rolled back events at each rollback occurrence, and because the elaboration of an event generates an average number of f_0 messages, a total number of $L_r f_0$ antimessages have to be send;

– the second term, which takes care of the invalid state cancellation time, is equal to $(L_r/x)d_d$; in fact only a fraction 1/x of rolled back events are checkpointed events;

– the third term, which takes care of the restoration time, is d_r;

– the fourth term, which takes care of the coasting forward time, given assumptions (2) and (3), is equal to $[(x-1)/2]d_c$ as in Ref. [15].

From the above considerations we get:

$$T_2 = L_r f_0 d_{out} + \frac{L_r}{x} d_d + d_r + \frac{x-1}{2} d_c \tag{6}$$

The term T_3 has the following structure:

$$T_3 = d_c + f_0 d_{out} \tag{7}$$

while, the term T_4 is

$$T_4 = d_s \tag{8}$$

Combining Eqs. (3)–(8) we finally obtain

$$T_{tot} = N_{tot}\left[(N_{in} + N_{ant})d_{in} + d_{sc} + P_r \times \left(L_r f_0 d_{out} + \frac{L_r}{x} d_d + d_r + \frac{x-1}{2} d_c\right) + d_c + f_0 d_{out} + \frac{1}{x} d_s\right] \tag{9}$$

From Eq. (9) we calculate the integer value x_{min} which minimizes T_{tot}:

$$x_{min} = \left\lceil \sqrt{\frac{2(d_s + P_r L_r d_d)}{P_r d_c}} \right\rceil \tag{10}$$

As in Refs. [12,15], the upper integer is chosen in Eq. (10) because empirical studies in Refs. [2,14] show that the execution time increases more quickly adopting checkpoint intervals that are smaller than the optimum compared to adopting checkpoint intervals greater than the optimum.

If the rollback frequency $P_r = 0$, then x_{min} becomes infinite. As already outlined in Refs. [14,15], this is not desirable because of the increase of the LP memory consumption to unacceptable levels, preventing fossil collection of any message.

As a last remark we note that if the parameter d_d is neglected, Eq. (10) becomes Eq. (1), exactly as in Ref. [15] (in fact, the rollback frequency P_r is the ratio between the number of rollbacks an the number of elaborated events, excluding the coasting forward ones).

4.2. The adaptive checkpointing algorithm

In this section we propose an adaptive checkpointing algorithm, based on the analysis performed in Section 4.1, for the optimization of the simulation execution time.

The steps for the algorithm are the following ones:

– as in Ref. [15], the simulation execution of each LP is divided into observation periods; the length of each period is L simulation events;

– the parameters P_r and L_r are measured at the end of each period;

– at the end of each period, by using a fixed number of most recent samples of P_r and L_r an extrapolation of the values that will characterize the next period is realized;

– the extrapolated values are introduced in Eq. (10) and the corresponding checkpoint interval is adopted in the next observation period.

Looking at Eq. (10), the two parameters that really drive the rollback behavior are $p_1 = P_r$ and the product $p_2 = P_r L_r$. Hence, in order to obtain smaller extrapolation errors, the extrapolation is applied directly on p_1 and p_2. The parameter p_1 must be in the range [0, 1] because it is a probability; the parameter p_2 must be in the range [0, 1) because of the following result:

Theorem 1

$$0 \leq P_r L_r < 1 \tag{11}$$

Proof (see Appendix A).

If values ranging out of the feasible domains are extrapolated for p_1 or p_2, then the adaptation is not realized and the old value of the checkpoint interval is still adopted.

As in Ref. [15], an upper limit x_{max} (equal to 15) on the value of x is introduced (because of the memory usage problem) so we have that, if introducing the extrapolated values of p_1 and p_2 in Eq. (10)

$$x_{min}(p_1, p_2) > x_{max} \tag{12}$$

then the value x_{max} is adopted in the next observation period. The following expression synthesizes the recalculation of x at the n-th observation period ($p_{1,n}$ and $p_{2,n}$

denote the extrapolated values of p_1 and p_2 at the n-th observation period):

```
**if** $n==0$ **then** $x_n = x_{initial}$
**else**{ $\langle$extrapolate $p_{1,n}$ and $p_{2,n}\rangle$;
    **if** $(0 \le p_{1,n} \le 1$ and $0 \le p_{2,n} < 1)$ **then** $x_n = \max(1, \min(x_{min}(p_{1,n}, p_{2,n}), x_{max}))$
    **else** $x_n = x_{n-1}$}
```

(13)

The proposed algorithm tries to *predict* the LP rollback behavior for the selection of the checkpoint interval. In fact the values of p_1 and p_2 measured in each period are never introduced in Eq. (10); they are used for the prediction of the rollback behavior parameters of the immediately following period.

The management of the algorithm carries a time-space overhead which depends on both the extrapolation technique and the number of samples used.

In the next Subsection we report the extrapolation technique we used to obtain the experimental results shown in Section 5.

4.2.1. Extrapolation technique

The basic formula we used in the extrapolation phase is Lagrange's formula. For the sake of the reader's convenience we report it here:

$$f(x) = \sum_{i=1}^{m} \frac{\prod_{l=1}^{m}(x - x_l)}{(x - x_i)\prod_{j=1, j \neq 1}^{m}(x_i - x_j)} f_i + E_m(x) \tag{14}$$

Each point x_i is an observation point of the function f to be extrapolated. Each value f_i is the corresponding sample. The point x is the one corresponding to the extrapolated value $f(x)$ and m is the number of used samples. Eq. (14) works both with regular and non-regular observation periods allowing the eventual adaptation of the period length in function of the extrapolation error $E_m(x)$. The experimental results in Section 5 have been obtained considering regular observation periods and a number of samples equal to five. This choice avoids the introduction of too much time-space overhead and, at the same time, the resulting algorithm improves performance.

In order to avoid a linear extrapolation, the algorithm starts to adapt the checkpoint interval when at least three samples are available for p_1 and p_2.

5. Performance evaluation

In this section we present experiments executed to evaluate performance of the adaptive checkpointing. The used simulation platform is SIMCOR [4]. This is a platform resulting from the Time Warp implementation on a hypercube multicomputer system (Intel iPSC/2). In SIMCOR aggressive message cancellation is used in cancellation phase and a Smallest-Timestamp-First scheduler is loaded on each processing element; furthermore, fossil collection is performed only when further memory, to store states or messages, is not available.

Experimental results show the correctness and the effectiveness of the adaptive checkpointing algorithm proposed in this paper, by means of a comparison with results obtained by using the algorithm in Ref. [15] (in Ref. [15] it has been already shown that adaptive checkpointing improves performance compared to static checkpointing so we do not report here experiments with static checkpointing).

We simulate two stochastic queuing models. The first model (M_1) is a fully connected net with 32 nodes where a constant message population circulates among the LPs. The timestamp increments are taken from an exponential distribution with mean 1 unit time and messages are equally likely forwarded to any other LP. The second model (M_2) is a modification of the first. It has 1 hot spot to which 10% of all messages are routed. In each simulation run the starting value for the checkpoint interval is $x_{initial}=5$ for each LP, and the number of messages in each node is 3.

We focus our attention on the execution time required for 150000 committed events. The first set of experiments (Figs. 4 and 5) has been realized varying the

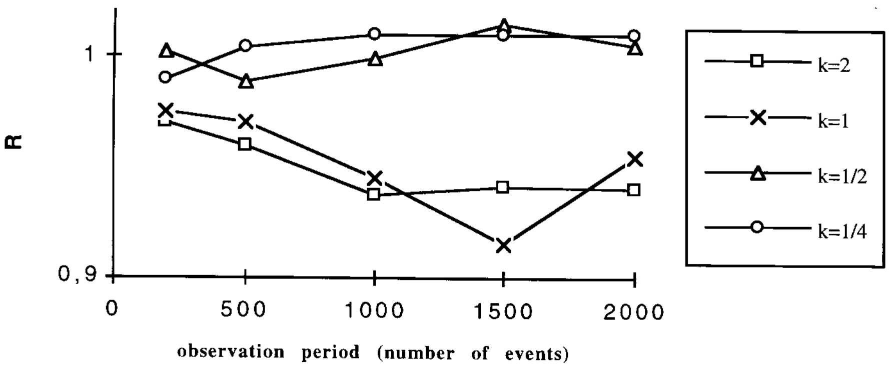


Fig. 4. Model M_1 – ratio R vs. the observation period length ($k=d_s/d_c$ and $d_d=0$).

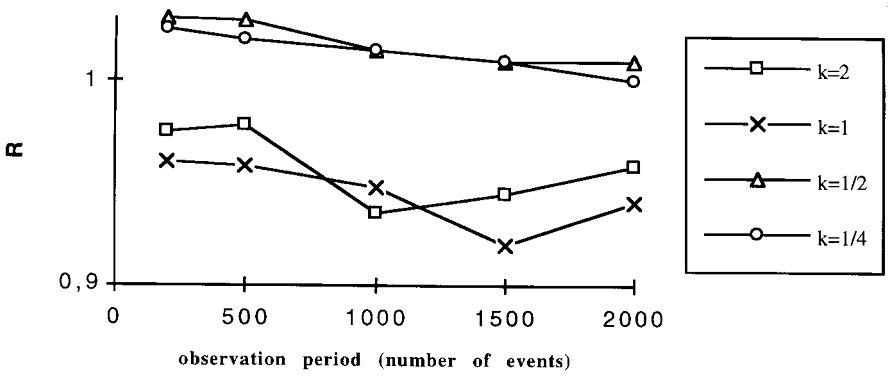


Fig. 5. Model M_2 – ratio R vs. the observation period length ($k=d_s/d_c$ and $d_d=0$).

ratio $k = d_s/d_c$ and with a negligible d_d value in order to effectuate an accurate comparison only between our adaptive methodology and the one in Ref. [15]. The second set (Fig. 6) has been carried out varying the ratio d_d/d_s to show the goodness of the analytical model in the case of non-negligible invalid-state cancellation cost. The ratio d_s/d_c and the ratio d_d/d_s are modified, as in Ref. [14], by introducing a variable delay loop into the event execution routine and into the state cancellation routine, respectively.

In Fig. 4 (model M_1) and in Fig. 5 (model M_2) the ratio R between the execution times obtained by our methodology and the one in Ref. [15] is shown vs. the length of the observation period (number of executed events) at the end of which adaptation is realized (the ratio R is evaluated by using the execution times resulting from the average of 20 runs).

The results in Figs. 4 and 5 show that our methodology improves performances for large values of the ratio $k = d_s/d_c$ (larger than or equal to 1 unit) while for small values of k (smaller than or equal to 1/2) it behaves like or a little worse than the one proposed in Ref. [15]. This phenomenon exactly matches the different adaptive approaches adopted. Looking with greater accuracy, let us consider Eq. (9) when $d_d = 0$, and evaluate the real value of x which minimizes it

$$x_{\min} = \sqrt{\frac{2}{P_r}\frac{d_s}{d_c}} \tag{15}$$

In order to study the sensitivity of x in function of the rollback probability we calculate the module of the first differential of Eq. (15) vs. the rollback probability P_r, i.e.:

$$\left|\frac{\partial x_{\min}}{\partial P_r}\right| = \left|\sqrt{\frac{1}{2P_r^3}\frac{d_s}{d_c}}\right| \tag{16}$$

From Eq. (16) it follows that, when the rollback probability changes, great changes of the x value are required mostly for large values of the parameter d_s/d_c (high sensitivity of $x_{\min}$ to the rollback probability when d_s/d_c is large). The methodology in Ref. [15] chooses the new checkpoint interval by using both its past value and the past rollback behavior. For this reason it performs very good adaptation in the

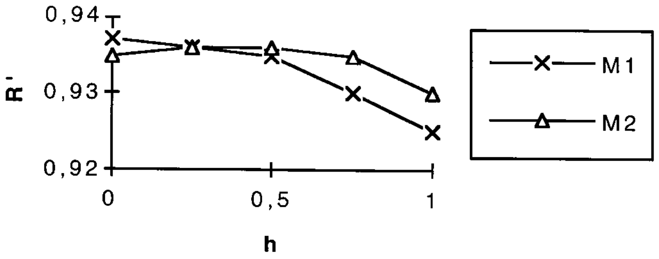


Fig. 6. Ratio $\mathbf{R}'$ vs. $h = d_d/d_s$ for both simulation models M_1 and M_2 ($k = 2$ and the observation period length is fixed at 1000 events).

case of small values of d_s/d_c (low sensitivity of x_{min} to the rollback probability) where small changes in the checkpoint interval are required even if the rollback behavior changes greatly. On the other hand, when d_s/d_c is large, it is more convenient (even if the extrapolation does not provide exact values for the rollback parameters) to base the choice of x on the prediction of the future rollback behavior, instead of using only information about the past.

We shall go on to propose some more experiments to show the goodness of our analytical model when the invalid-state cancellation time d_d is not negligible. In Fig. 6 the ratio $\mathbf{R}'$ between the execution times (average values of 20 runs) obtained by our methodology and the one in Ref. [15] is shown vs. the ratio $h = d_d/d_s$ (the observation period length is fixed at 1000 events and the ratio $d_s/d_c = 2$). The curves in Fig. 6 show that, when the ratio h increases, the performance obtained by our methodology, compared to the one obtained by the methodology in Ref. [15], increases slightly. This is because when h increases, the error in selecting the checkpoint interval using Eq. (1), instead of Eq. (10), grows slightly.

6. Conclusions

This paper addresses the problem of defining the relation between the execution time of a logical process in a Time Warp simulation, and the value of its checkpoint interval. As first step, we introduce an analytical model describing the overall checkpointing-rollback overhead as a function of the checkpoint interval of a logical process. The model shows the value of the checkpoint interval which minimizes such an overhead is a function of the rollback behavior of the logical process. Usually, this behavior does not reach a steady state, thus, we also introduce an adaptive methodology for dynamically adjusting the checkpoint interval according to the changes in the rollback behavior. The methodology exploits information about the past rollback behavior of a logical process for predicting the future values of the rollback parameters, which are used for dynamically selecting the value of the checkpoint interval. Experimental results show that, in some simulation scenarios, our methodology improves performance compared to already existing ones.

Appendix A

Theorem 1

$$0 \leq P_r L_r < 1 \tag{A.1}$$

Proof

Right part.

Let N_{com} be the number of committed events executed by the LP and N_{tot} be the total number of executed events (i.e., the number of executions of the loop in Fig. 3). Under assumptions (1) and (3) the following equation holds:

$$N_{tot} = N_{com} + N_{roll} \tag{A.2}$$

where N_{roll} is the number of rolled-back events during the parallel execution of the simulation. Eq. (A.2) shows that, in batch execution, coasting forward events do not affect the number of execution loops. The relationship between N_{com} and N_{tot} can be expressed as follows:

$$N_{tot} = \frac{N_{com}}{1 - P_r L_r} \tag{A.3}$$

In fact, under assumptions (1) and (3), the number of rolled-back events is given by the total number of rollback executions ($P_r N_{tot}$) multiplied by the rollback length, i.e.

$$N_{roll} = [P_r N_{tot}] L_r \tag{A.4}$$

and from Eq. (A.2)

$$N_{roll} = [P_r (N_{com} + N_{roll})] L_r \tag{A.5}$$

therefore

$$N_{roll} = \frac{P_r L_r}{1 - P_r L_r} N_{com} \tag{A.6}$$

and Eq. (A.3) is easily obtained combining Eqs. (A.2) and (A.6). The constraint for the consistency of Eq. (A.3) is

$$P_r L_r < 1 \tag{A.7}$$

in fact, if inequality Eq. (A.7) does not hold, the simulation reaches one of the following inconsistent states:

(1) if $L_r = 1/P_r$ the LP is locked in virtual time, resulting in an infinite number of total parallel events to be executed;

(2) if $L_r > 1/P_r$ the LP is (theoretically) going back in virtual time, resulting in a negative number of total parallel events to be executed.

Left part.

Denoting with k_{obs} the number of rollbacks we can write

$$L_r = \frac{N_{roll}}{k_{obs}} \tag{A.8}$$

but at least one event is undone because of a rollback so $N_{roll} \geq k_{obs}$, thus $L_r \geq 1$. Given that $P_r \geq 0$, then $P_r L_r \geq 0$. Q.E.D.

References

[1] H. Bauer, C. Sporrer, Reducing rollback overhead in Time Warp based simulation with optimized incremental state saving, in: Proceedings of the 26-th Annual Simulation Symposium, 1993, pp. 12–20.

[2] S. Bellenot, State skipping performance with the Time Warp operating system, in: Proceedings of the 6-th Workshop on Parallel and Distributed Simulation, Newport Beach, CA, 1992, pp. 33–42.
[3] K.M. Chandy, J. Misra, Distributed simulation: A case study in design and verification of distributed programs, IEEE Trans. Software Eng. SE5 (5) (1979) 213–224.
[4] B. Ciciani, M. Angelaccio, An interface to develop Time-Warp based parallel simulations, in: Proceedings of the Massively Parallel Processing Conference, Delft, Holland, 1994, pp. 20–21.
[5] J. Fleischmann, P.A. Wilsey, Comparative analysis of periodic state saving techniques in Time Warp simulators, in: Proceedings of the 9-th Workshop on Parallel and Distributed Simulation, Lake Placid, NY, 1995, pp. 50–58.
[6] R.M. Fujimoto, Parallel discrete event simulation, Commun. ACM 33 (10) (1990) 30–53.
[7] A. Gafni, Space management and cancellation mechanisms for Time Warp, Technical Report 85-341, University of Southern, California, 1985.
[8] D. Jefferson, H. Sowizral, Fast concurrent simulation using the Time Warp mechanism; part I: local control, Technical Report N1906AF, RAND Corporation, Santa Monica, CA, 1982.
[9] D. Jefferson, Virtual time, ACM Trans. Programming Languages Syst. 7 (3) (1985) 404–425.
[10] D. Jefferson, B. Beckman, F. Wieland, L. Blume, M. DiLoreto, P. Hontalas, P. Laroche, K. Sturdevant, J. Tupman, V. Warren, J. Wedel, H. Younger, S. Bellenot, Distributed simulation and the Time Warp operating system, in: Proceedings of the 12-th Symposium on the Operating Systems Principles, New York, 1987, pp. 77–93.
[11] Y.B. Lin, B.R. Preiss, W.M. Loucks, E.D. Lazowska, Selecting the checkpoint interval in Time Warp simulation, in: Proceedings of the 7-th Workshop on Parallel and Distributed Simulation, San Diego, CA, 1993, pp. 3–10.
[12] A.C. Palaniswamy, P.A. Wilsey, An analytical comparison of periodic checkpointing and incremental state saving, in: Proceedings of the 7-th Workshop on Parallel and Distributed Simulation, San Diego, CA, 1993, pp. 127–134.
[13] A.C. Palaniswamy, P.A. Wilsey, Adaptive checkpoint intervals in an optimistically synchronized parallel digital system simulator, in: Proceedings of the IFIP TC/WG10.5 International Conference on Very Large Scale Integration, 1993, pp. 353–362.
[14] B.R. Preiss, W.M. Loucks, D. MAcIntyre, Effect of the checkpoint interval on time and space in Time Warp, ACM Trans. Modeling Comput. Sim. 4 (3) (1994) 223–253.
[15] R. Ronngren, R. Ayani, Adaptive checkpointing in Time Warp, in: Proceedings of the 8-th Workshop on Parallel and Distributed Simulation, Edinburgh, Scotland, 1994, pp. 110–117.
[16] D. West, K. Panesar, Automatic incremental state saving, in: Proceedings of the 10-th Workshop on Parallel and Distributed Simulation, Philadelphia, PA, 1996, pp. 78–85.