

# On the Selection of the State Saving Strategy In Time Warp Parallel Simulations

Hussam M. Soliman

College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia; E-mail: hsoliman@ccis.ksu.edu.sa

*The Time Warp parallel simulation algorithm requires that each logical process save its state for later recovery at the time of a rollback. As the simulated model may be composed of various subsystems with contrasting and dynamically changing characteristics, it may be more efficient to have each subsystem choose the state-saving technique which is more suitable to its observed runtime behavior. In this paper, a state-saving technique is proposed which automatically chooses either periodic or incremental state saving for each logical process based on a simple decision criterion. An experimental study is also conducted to evaluate the performance of the proposed technique.*

**Keywords:** Parallel simulation, checkpointing, performance evaluation, time warp algorithm

## 1. Introduction

Parallel and distributed simulation techniques are divided into two broad classes: *conservative* and *optimistic*. In conservative techniques, a logical process's clock can only be advanced when it can be ensured that causality constraints are not violated. This guarantees correct order for all event processing but requires the processes to *look ahead* in simulated time in order to avoid deadlock. On the other hand, in optimistic techniques, a process's clock may be advanced until a violation of causality constraints is discovered. Simulation time must then be rolled back to a consistent state. For example, in the Time Warp optimistic technique, a rollback mechanism is initiated in each logical process (LP) if a violation of causality constraints is discovered in order to cancel the effects of the incorrect computations and restore the most recent valid state. This, however, requires that the state of the logical process be saved at regular intervals, which adds additional overheads, regardless of whether or not rollbacks occur.

One approach to reducing state saving overheads is to use special-purpose hardware [1]. Another approach is to have each logical process checkpoint its state at periods greater than one. Normally, if a process with a local virtual time (LVT) equal to  $T_3$  receives an event message with timestamp  $T_2 < T_3$ , the process normally rolls back to  $T_2$  and resumes execution from that point on. In this case, however, if  $T_1 < T_2$  is the timestamp of the last checkpointed event, the events with timestamps between  $T_1$  and  $T_2$  are re-executed to produce the state of the process at  $T_2$ . This re-execution phase is called *coasting forward*. In this phase, any scheduling of future events is ignored because the purpose of this phase is to restore a process state that was not

preserved. The selection of the checkpoint interval is a tradeoff between the total cost of checkpointing and the amount of re-execution in coasting forward. As the checkpointing interval is increased, both the space and time overheads are decreased and the cost of coasting forward increases, and vice versa.

Several techniques for dynamically finding the optimal checkpointing interval for Time Warp have been reported. Lin et al. [2] present a mathematical model of periodic checkpointing and use it to develop a dynamic checkpoint interval selection algorithm which takes into account the effect of the checkpoint interval on rollback behavior. Ronngren and Ayani [3] propose an adaptive technique for the determination of the optimal checkpoint interval at runtime by calculating the checkpoint interval for the current observation period as a weighted sum of the previous approximation and the calculated optimal interval. This allows the checkpoint interval to adapt to changes in rollback behavior. Fleischmann and Wilsey [4] present a heuristic algorithm for dynamically adjusting the checkpoint interval in Time Warp simulations of logic circuits. Their algorithm uses a cost function which equals the sum of state saving and coast-forward overheads, and is evaluated over regular observation periods. Preiss, Loucks and Macintyre [5] conduct an experimental study of the effects of the checkpoint interval on Time Warp behavior and total execution time for closed stochastic queuing networks connected with different topologies and using different cancellation strategies and scheduling algorithms.

In applications with large state sizes, an alternative to periodic checkpointing is to save the change in state between successive event executions, called a state *increment*. To reconstruct an old state from the current state, the saved state increments are added up until the desired state is reached. This is the main idea of the incremental state saving technique [6]. This technique has the advantages of low memory consumption and low state saving time, provided the saved state increments are relatively

Received: December 1997; Revised: September 1998; Accepted: September 1998

TRANSACTIONS of The Society for Computer Simulation International  
ISSN 0740-6797/99  
Copyright © 1999 The Society for Computer Simulation International  
Volume 16, Number 1, pp.32-36

small. Planaiswamy and Wilsey [7] give a comparative analysis of periodic checkpointing and incremental state saving. They conclude that incremental state saving has the potential to outperform periodic state saving if the average number of state increments saved after each event execution is small, the event execution time is relatively large, and the rollback distance is small. Ronngren et al. [8] investigate the performance of several periodic state saving and incremental state saving mechanisms. Their empirical study indicates that incremental state saving is preferred when the state sizes are large and only a small fraction of the state vector is modified in an event execution. The study also indicates that the best state saving method depends mainly on the simulation model characteristics in a way that is usually not obvious to the user.

Currently, the selection of the appropriate state saving mechanism for the simulated model is the responsibility of the simulation programmer, who may not have enough *a priori* knowledge of the model runtime characteristics. It is, therefore, desirable to have the simulation kernel dynamically select the best state saving strategy for each logical process in the model based on runtime behavior. This can be done if the following two conditions are satisfied: (1) the state saving mechanisms are made transparent to the user, and (2) methods are developed to guide the automatic selection process. Periodic state saving can easily be made transparent to the user. In addition, the cost to achieve transparency for incremental state saving is usually negligible for many applications [9]. This would require the insertion of code to save the state before the modification of any state variables. This paper presents a method, based on an analytical model of state saving in Time Warp, which allows the Time Warp kernel to dynamically select the state saving mechanism which best suits the observed runtime characteristics of each logical process.

## 2. Selection Criteria for State Saving Mechanism

In the following, we adopt the analytical model developed in [2] for state saving in Time Warp. In particular, we assume that the lifetime of each logical process consists of  $k$  rollback cycles. We further assume that each rollback cycle consists of three phases: a coast forward phase (in the case of periodic state saving) of distance  $\gamma_i$  events, a forward execution phase of distance  $\alpha_i$  events, and a rollback phase of distance  $\beta_i$  events. Figure 1 illustrates the structure of a rollback cycle. The time taken to save (or restore) a whole LP state,  $\delta_s$ , the time taken to save (or

restore) a state increment,  $\delta_{inc}$ , and the time taken to execute a single event,  $\delta_e$ , are all assumed to be constant throughout the simulation.

In order to derive decision criteria for state saving method selection, cost models are developed for periodic state saving (PSS) and incremental state saving (ISS). In the  $i$ th rollback cycle, the state saving cost of PSS consists of the cost of saving the entire state for  $\alpha_i/X$  events in the forward execution phase, the cost of coasting forward  $\gamma_i$  events, and the cost of restoring one old state. In mathematical terms:

$$CostPSS = \sum_{i=1}^k \frac{\alpha_i \delta_s}{X} + \gamma_i \delta_e + \epsilon_i \delta_s$$

where  $\epsilon_i = 0$  if  $k = 1$  and  $\epsilon_i = 1$  if  $k > 1$ . On the other hand, the state saving cost of ISS in the  $i$ th rollback cycle consists of the cost of saving the state increments for  $\alpha_i$  forward events and the cost of retrieving the state increments for  $\beta_i$  events at the time of a rollback. Therefore:

$$CostISS = \sum_{i=1}^k \alpha_i \delta_{inc} + \beta_i \delta_{inc}$$

The dynamic state saving method depends on monitoring the execution of each Time Warp logical process during observation intervals of length  $N$  event executions. At the end of each interval, the cost function of the currently used state saving technique is re-evaluated using collected statistics on the number of rollbacks and the number of forward-executed, rolled-back and coast-forward events in each rollback cycle. A decision on which state saving technique to use in the next interval is then made using the newly calculated cost value of the currently used state saving method and the last cost value calculated for the alternative method, as follows:

- 1- switch to PSS if  $CostISS$  during the current interval  $>$   $CostPSS$  during the last observation interval in which PSS was used;
- 2- switch to ISS if  $CostPSS$  during the current interval  $>$   $CostISS$  during the last observation interval in which ISS was used.

The above selection criteria ensure that the current state saving method will not be changed unless the cost in the current observation interval turns out to be strictly greater than the last reported cost for the other state saving method. This avoids ex-

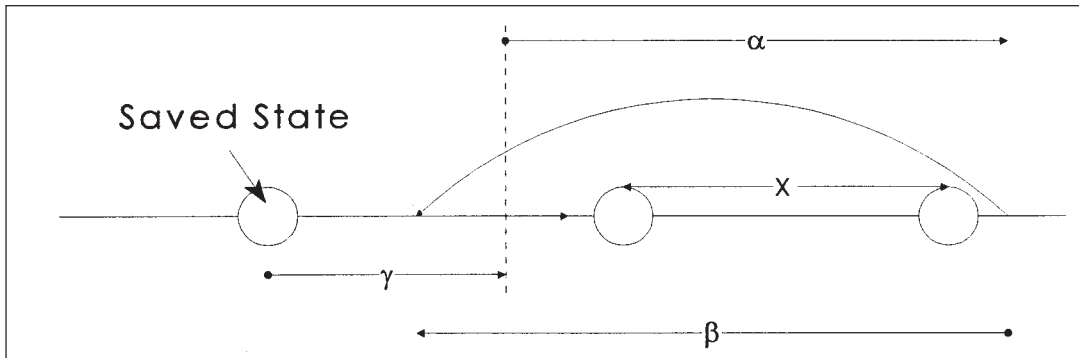


Figure 1. The structure of a rollback cycle

cessive or premature switching of state saving methods. Only in the cases where the cost of the current method exceeds that of the alternative method will a change of methods be justified, as it may reflect a change in the characteristics of the underlying logical process.

Switching from one state saving method to another involves changing the size of the saved portion of the state (a whole state in PSS or a state increment in ISS). In addition, it involves changing the frequency of state saving (every  $X$  event in PSS or every event in ISS), and also changing the rollback recovery procedure executed after the occurrence of a rollback (a coasting forward mechanism in PSS or state increment addition in ISS). Both state saving methods must be implemented simultaneously in the Time Warp kernel in a way that permits efficient switching between them. The implementation must also allow for multiple rollback recovery mechanisms in cases where one rollback spans multiple observation intervals in which different state saving methods were used. In practice, however, it is expected that the observation interval length will be much larger than the maximum rollback length for most applications.

### 3. Performance Evaluation

An implementation of Time Warp is developed on a network of eight Sun 1+ workstations interconnected by a 10Mbit/sec Ethernet. A static process scheduler was selected using a smallest-timestamp-first scheduling policy. A fast version of the Chandy-Lamport snapshot algorithm is used for GVT calculation [10]. Each of the eight machines runs 16 logical processes connected as a  $4 \times 4$  torus, forming an overall  $8 \times 16$  torus topology. A synthetic workload simulation with a shifted exponential service function is used in all experiments. Each logical process selects the output link on which it forwards the next outgoing message according to a uniform distribution. The event-execution delay, the state saving delay, and the state-increment-saving delay are all implemented as pure delay loops whose delay values are parameters of the experiments. All logical processes are initially populated with 20 event messages.

Two simulation load parameter sets describing different load characteristics with respect to state saving are considered. The parameters of the two loads are shown in Table 1. Parameter set S1 is intended to favor the use of ISS, and parameter set S2 is intended to favor the use of PSS. Each processor is assumed to run a subsystem of the simulated model, consisting of 16 logical processes, possessing characteristics described by one of the two load parameter sets. Three loading cases are considered in the conducted experiments, as shown in Table 2. In each case, the average state saving cost on each processor is calculated from collected statistics when the processor runs Time Warp with only PSS, Time Warp with only ISS, and Time Warp with the

Table 1. Load parameter sets (in milliseconds)

Parameter Set	Event Execution Delay ( $\delta_e$ )	State-Saving Delay ( $\delta_s$ )	Increment Saving Delay ( $\delta_{ind}$ )
Set 1 (S1)	0.72	2.000	0.147
Set 2 (S2)	1.04	0.147	0.085

dynamic state saving method (DSS). For both PSS and DSS, the adaptive checkpointing technique developed in [3] is implemented and used for dynamic determination of the state saving interval  $X$  every 100 event executions with  $\rho = 0.4$ .

Figures 2 through 4 show the average state saving costs per processor for each loading case. It can be seen in Figure 2 that the per-processor cost of using DSS in processors P1–P4, which carry S1 loads, was close to that of ISS. For P2 and P3, the cost of DSS was slightly higher than that of ISS. This is explained by the overhead incurred by initially operating with the *wrong* state saving method for a number of observation intervals until the cost of this method exceeds a threshold, after which the algorithm corrects the situation. For processors P5–P8, which carry S2 loads, the cost of DSS is close to that of PSS.

Similar observations are made in Figure 3, where odd-numbered processors carry S1 loads and even-numbered processors carry S2 loads. DSS performance on each processor is close to that of the state saving method favored by the load carried by that processor. Namely, for odd-numbered processors, the cost of DSS is close to that of ISS, and for even-numbered processors, the cost of DSS is close to that of PSS. In Figure 4, all the eight processors are loaded with S1 loads. It can be seen that DSS performance is close to that of ISS on all processors.

Figure 5 shows the effect of using DSS on the total simulation execution time for each of the three loading cases. In the first two loading cases the Time Warp kernel which implements DSS was able to complete the simulation execution in less time than kernels which use PSS or ISS. This is explained by the fact that it automatically selects for the logical processes on each processor the most suitable state saving method for the load which it carries. This is not the case for the PSS and ISS Time Warp kernels which employ one state saving method irrespective of the dynamic properties of the simulated model, which results in higher time costs.

In the third loading case, the ISS Time Warp kernel achieved the lowest execution time because, in this case, all logical processes carry the same uniform load, which favors the use of ISS. The DSS kernel, however, recognizes this situation only after consuming some time using the wrong state saving method in a number of initial observation intervals. This DSS overhead

Table 2. Loading cases for the eight processors

Processor	P1	P2	P3	P4	P5	P6	P7	P8
Case1	S1	S1	S1	S1	S2	S2	S2	S2
Case2	S1	S2	S1	S2	S1	S2	S1	S2
Case3	S1	S1	S1	S1	S1	S1	S1	S1

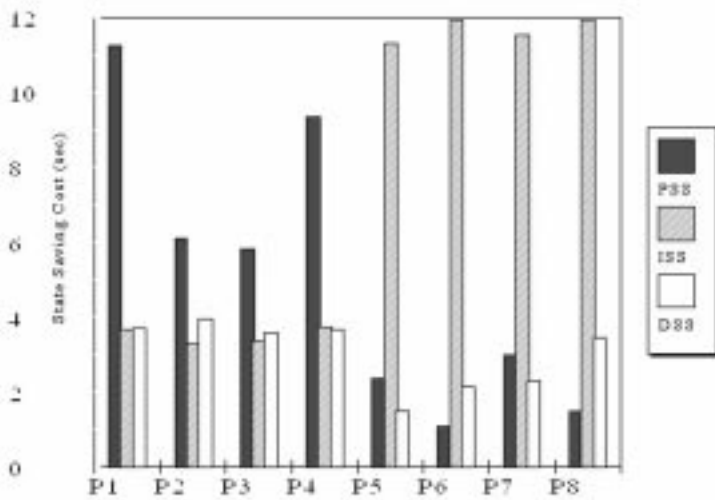


Figure 2. State saving cost per processor for loading case 1

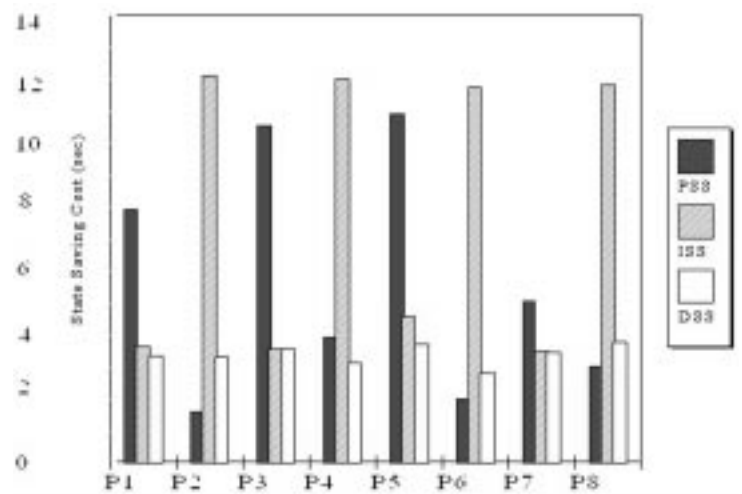


Figure 3. State saving cost per processor for loading case 2

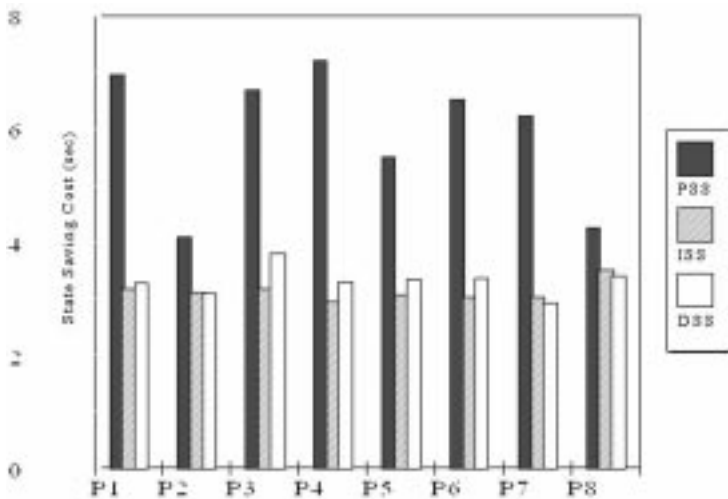


Figure 4. State saving cost per processor for loading case 3

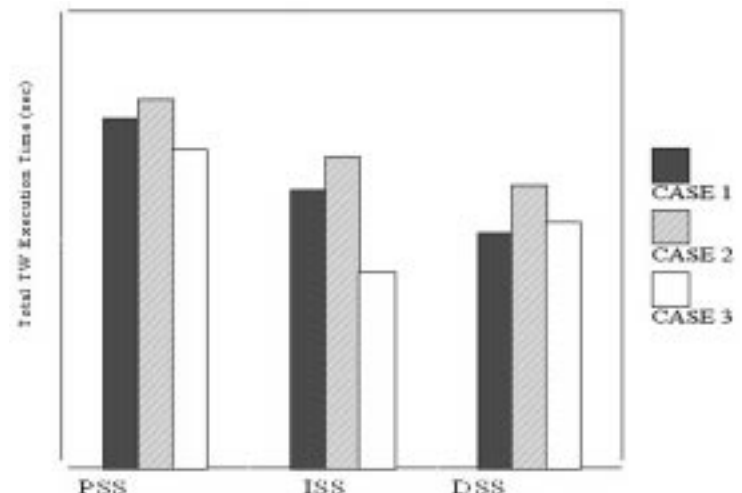


Figure 5. Total execution time for the three loading cases

is less, however, than the overhead of using the wrong state saving method throughout the simulation, as is the case of the PSS kernel.

**4. Conclusion**

A dynamic state saving technique for Time Warp was proposed and implemented. The technique dynamically chooses either the periodic or incremental state saving method, every observation period, for each logical process based on the values of analytically derived cost functions and simple decision criteria. An experimental study was also conducted on a network of workstations to evaluate the performance of Time Warp when using this technique compared to that of Time Warp which uses periodic or incremental state saving, for both uniform and nonuniform loading conditions. Experimental results show that the per-processor cost of using DSS was very close to the cost of the state saving method which the loading conditions favor on each processor. The advantage of using DSS is clearly demonstrated by the total simulation execution time results. It was seen that using DSS not only relieves the simulation programmer from

the responsibility of choosing the appropriate state saving method for the simulated model, which can even have dynamically changing characteristics, but also reduces the total execution time of the simulation. This is because it chooses, after a number of observation intervals, the state saving method which best suits the dynamic properties of each logical process in the simulated model.

In the future, it remains to study the effect of the observation period length on the performance of the proposed scheme, and to find ways to optimize its selection. Programming interface issues will also be addressed in more detail.

**5. References**

[1] Fujimoto, R.M., Tsai, J., Gopalakrishnan, G. "Design and Evaluation of the Rollback Chip: Special Purpose Hardware for Time Warp." *IEEE Transactions on Computers*, Vol. 41, pp 68-82, 1992.

[2] Lin, Y.B., et al. "Selecting the Checkpoint Interval in Time Warp Simulation." *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, pp 3-10, 1993.

- [3] Ronngren, R., Ayani, R. "Adaptive Checkpointing in Time Warp." *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, pp 110-117, 1994.
- [4] Fleischmann, J., Wilsey, P.A. "A Comparative Analysis of Periodic State Saving Techniques in Time Warp Simulators." *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, pp 50-58, 1995.
- [5] Preiss, B.R., Loucks, W.M., Macintyre, I.D. "Effects of the Checkpoint Interval on Time and Space in Time Warp." *ACM Transactions on Modeling and Computer Simulation*, Vol. 4, pp 223-253, 1994.
- [6] Bauer, H., Sporrer, C. "Reducing Rollback Overhead in Time-Warp Based Distributed Simulation with Optimized Incremental State Saving." *Proceedings of the 26th Annual Simulation Symposium*, pp 12-20, 1993.
- [7] Palaniswamy, A.C., Wilsey, P.A. "An Analytical Comparison of Periodic Checkpointing and Incremental State Saving." *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, pp 127-134, 1993.
- [8] Ronngren, R., et al. "Comparative Study of State Saving Mechanisms for Time Warp Synchronized Parallel Discrete Event Simulation." *Proceedings of the 29th Annual Simulation Symposium*, pp 5-14, 1996.
- [9] Ronngren, R., et al. "Transparent Incremental State Saving in Time Warp Parallel Discrete Event Simulation." *Proceedings of the 10th Workshop on Parallel and Distributed Simulation*, pp 70-77, 1996.
- [10] Soliman, H.M., Elmaghaby, A.S. "An Improved Chandy-Lamport Snapshot Algorithm for GVT Approximation in Distributed Simulations." *Proceedings of the 8th International Conference on Parallel and Distributed Computing Systems*, pp 473-477, 1995.

## 6. Additional Reading

- Bellenot, S. "State Skipping Performance with the Time Warp Operating System." *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, pp 53-61, 1992.
- Cleary, J., et al. "Cost of State Saving & Rollback." *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, pp 94-101, 1994.



**Hussam M. Soliman** earned a PhD in 1995 from the University of Louisville. He is now an Assistant Professor and Chairman of the Information Systems Department at the College of Computer and Information Sciences, King Saud University, Saudi Arabia. His research interests include parallel and distributed simulation, object-oriented modeling and simulation, and decision support systems. He is a member of ACM, IEEE, SCS, ISCA, Eta Kappa Nu and Tau Beta Pi.