Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

EVENT SENSITIVE STATE SAVING IN TIME WARP PARALLEL DISCRETE EVENT SIMULATIONS

Sven Sköld
Robert Rönngren

Simulation Laboratory
Department of Teleinformatics
Royal Institute of Technology
Stockholm, SWEDEN

ABSTRACT

In this paper we present *event sensitive state saving* as a novel idea for sparse state saving in Time Warp synchronized parallel discrete event simulations. This state saving model is aimed at efficient simulation of models where the execution time or granularity for different types or classes of events typically has a large variance. The event sensitive state saving mechanism is sensitive to which class of event the previously executed event belongs, and decide when to save simulation state based on this information. We present an analytical analysis and compare this new state saving model with the performance of an existing adaptive state saving model as well as the traditional Time Warp state saving model with fixed state saving interval.

1 INTRODUCTION

The most well known optimistic parallel discrete event simulation mechanism is Time Warp, proposed by (Jefferson 1985). Any parallel simulation mechanism should guarantee that the simulation is synchronized such that the output is free from effects generated by causality errors. Time Warp uses a detection and recovery mechanism to address these issues. In Time Warp, a causality error is detected when a timestamped message arrives at a logical process (LP) and the timestamp of the message, i.e. the message receive time, is less than the time of the last processed event at that LP. The event that causes the causality error is called a straggler event. To recover from an erroneous state, the LP is rolled back in time to a state before the straggler event. Before resuming the execution, any possible effects from all messages that have been sent with timestamps greater than the straggler timestamp need to be undone. This is achieved by resending negative versions of previously sent messages, so called anti-messages, which will annihilate the original messages and thus undo the effect from the original messages. However, these anti-messages might possibly act as straggler messages themselves and hence cause further rollbacks at other LPs. Despite what may appear as a complicated mechanism, Time Warp has proved to be a promising mechanism for parallel simulation (Fujimoto 1990). However, there exist some performance issues related to Time Warp which still merits further attention by the research community.

One potential performance bottleneck of Time Warp is the need to periodically save the state for each LP to later be able to roll back to earlier states. Without an effective memory manager combined with an efficient state saving method, the amount of time as well as memory needed to save states could severely degrade performance. Furthermore, it might also put a limit on the maximum size of the models possible to simulate due to excessive memory consumption. During the last decade, two methods to reduce the state saving overhead has been proposed: incremental state saving (Bauer, and Sporrer 1993) (Palaniswamy, and Wilsey 1993) and sparse or infrequent state saving (Lin et al. 1993). In this paper we focus on sparse state saving methods.

Traditionally in Time Warp simulations, the state or state variables are saved after or before the processing of each event at an LP. This is usually referred to as *copy state saving*. However, one can observe that a state can be restored by reloading an earlier state and re-executing or *coasting forward* all events in between. Thus one could potentially save execution time and memory by not saving the state after each event, but instead save the state of an LP at some specific time interval which is greater than one - hence the name infrequent or *sparse state saving*.

Several methods for sparse state saving has been proposed. To our best knowledge these models are, without exceptions, based on the assumption that the execution time of events in coast forward can be approximated with a mean execution time, i.e. with an underlying assumption that the execution time, or granularity, show little variance. However, there exist many important classes of

simulations where this obviously is not a good or even valid assumption. Consider for example battlefield simulations or simulations of mobile communication systems where many events are simple position updates for mobile entities. These events in general have very low granularity. However, these models typically also feature very large grain events which occur when entities in battlefield simulations enter into combat or when mobile units in a personal communication system perform handovers or interference measurements.

In this paper we propose an *event sensitive state saving* method for sparse state saving. This method takes into consideration the granularities of different event types or classes in the simulation. We present a simple analysis and compare this new state saving method to the performance of an adaptive state saving method (Rönngren, and Ayani 1994) as well as a traditional Time Warp simulator with a fixed state saving interval. We use two simple mobile communication network models as benchmarks.

This paper is organized as follows: Section 2 gives a short introduction to sparse state saving in Time Warp; other related work in this area is summarized in Section 3; Section 4 describes the event sensitive state saving approach; Section 5 shows our experimental results with this new state saving model; and finally Section 6 concludes the paper and summarizes this work.

2 SPARSE STATE SAVING

Sparse or infrequent state saving approaches are based on the observation that one simulation state can be recreated from an earlier state by re-executing the events between the two states. Thus, the simulation state does not need to be saved or checkpointed after the processing of each event. Hence, we can trade the overhead of saving state after each event for the overhead of restoring a state by coasting forward, or re-executing, the events in between the latest previously saved state and the current state (Preiss, Loucks, and MacIntyre 1994). For state restoration purposes and to assert the simulation can progress it is sufficient that at least one earlier state before the straggler event has been saved as well as all subsequent events that have been processed but not checkpointed, see the simulation snap shot example in Figure 1.

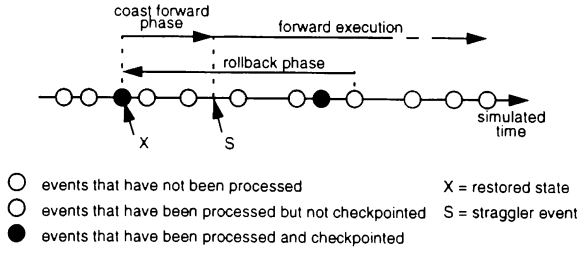

Figure 1: Simulation Snap Shot with $\chi = 5$

When a straggler event occurs, the saved state closest before the straggler event in simulation time (X in Figure 1) is recovered and the simulation coasts forward (without re-sending any messages) to the time of the straggler (S in Figure 1) where normal forward execution is resumed. We can therefore divide the simulation progress cycle of an LP into three different phases (Lin et al. 1993): forward execution phase; rollback phase; and coast forward phase.

If the frequency of state saving is reduced, the time to save states is also reduced and hence the cost for the forward execution phase is reduced. However, this reduction is at the expense of possibly longer rollbacks and of longer coast forward phases which increase the state restoration time. An appropriate state saving interval in a sparse state saving method should obtain a low cost from the forward execution phase while maintaining a low cost for the rollback and coast forward phases. Hence, this choice is dependent on the relations between the time to save the state of an LP; the time to coast forward or re-execute an event; the number of rollbacks; as well as the number of events involved in a rollback.

3 RELATED WORK

As the use of the Time Warp mechanism for optimistic synchronization of parallel discrete event simulations has increased, many researchers have presented different sparse state saving methods which differ in how to select the state saving interval. The approaches can be divided into two groups: *fixed sparse state saving*, where the chosen state saving interval is kept constant during the entire simulation, and *adaptive sparse state saving*, where the state saving interval is dynamically adapted to changes in monitored simulator parameters.

3.1 Fixed Sparse State Saving Method

In the simplest method, all LPs use the same static state saving interval, i.e. the simulation state for each LP is always saved after every χth event execution, where χ is the state saving interval. These methods work best when the simulation model is homogeneous, i.e. all LPs are identical and do not change behavior over time. The question is how to select an optimal state saving interval χ. (Lin et al. 1993) developed a manual regression model which works well for homogeneous models.

3.2 Adaptive Sparse State Saving Methods

(Fleischmann, and Wilsey 1995) performed an empirical study on sparse state saving approaches and present a heuristic algorithm for adapting to changes in the simulation model execution behavior. This method measures the

time to execute N events and compares this measurement with the previous measurement. If the execution time has increased "significantly", the state saving interval is decreased by one otherwise it is increased by one.

(Rönngren, and Ayani 1994) developed an adaptive state saving method where the simulator dynamically changes the state saving interval for each LP depending on the individual LP's rollback behavior. Using this approach, a near optimum choice of the state saving interval can be selected at run time. For each LP, the monitored parameters are the number of processed events R_{obs} (including events that have been rolled back) and the number of rollbacks k_{obs} during an observation period. It is then possible to calculate the state saving interval χ_{min}, see Equation (1), which minimizes the sparse state saving overhead for the LP. In this equation, δ_s is the mean time needed to save the simulation state and δ_c is the mean execution time for an event in the coast forward phase. Equation (1) is then used to iteratively refine χ while adapting to changes in the rollback frequency R_{obs} / k_{obs}.

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

Recent work (Rönngren et al. 1996) shows that this adaptive method gives the best results when comparing the state saving performance for various sparse state saving techniques when simulating large, realistic cellular communication systems.

4 EVENT SENSITIVE STATE SAVING

We observe that previous work in estimating the optimal state saving interval typically depend on an estimation of the *mean* time for execution or coast forward of an event (δ_e or δ_c) or the estimation of the time to save state and time to coast forward (δ_s / δ_c) ratio. However, for these estimations to be optimal the variance should not be large, i.e. all types of events should have similar performance when comparing execution and coast forward times. Intuitively, even if there exist simulation models where the event execution and coast forward time variations are negligible, this can not generally be true. Clearly the coast forward time could be reduced if it was possible to reduce the probability of having to coast forward over large granular events. We therefore claim that this variance in event characteristics is an important factor when selecting state saving interval in a Time Warp synchronized discrete event simulation.

If we divide all possible types of events in a simulation model into a set of event classes, where members of each class exhibit a similar behavior when it comes to execution and coast forward times, we can treat each class as "individual events" without loosing generality. For a general simulation model the execution time for different event classes in an LP might vary considerably. We therefore propose a new sparse state saving method, called *event sensitive state saving* (ESSS), where we take this variance into consideration when selecting the optimal state saving interval.

4.1 The ESSS Approach

To further reduce the state saving overhead in sparse state saving methods we target the coast forward phase in the simulation progress cycle (see Figure 1). The time to execute this phase depends on the length of the state saving interval χ as well as the individual coast forward times for the events in the coast forward phase. The mean number of events in the coast forward phase is $(\chi\text{-}1)/2$ under the assumption that state saving points and stragglers are independent. If we denote the *mean* coast forward time for all event classes as δ_c the average state saving overhead due to one coast forward phase is estimated as:

$$T_{coh} = \delta_c (\chi - 1) / 2 \quad (2)$$

In traditional sparse state saving models, the state of an LP is saved after or before the execution of *any* event in a simulation run. From the coast forward point of view, this might not be the best approach. If the model under simulation is described by events where the coast forward time for different events has high variance, it should be more efficient to save the state after events that is *expensive*, i.e. with long coast forward times, compared to after events with short coast forward times. By employing a state saving technique that favors state saving after expensive events we could reduce the mean δ_c, i.e. it would be possible to reduce the *effective* δ_c that will add to the coast forward overhead T_{coh} in Equation (2).

The goal of the ESSS approach is therefore to implement a sparse state saving model where state saving is more probable after expensive high granularity events, than after events with lower granularity.

4.2 Execution Time Model

To understand the effects on the state saving overhead in sparse state saving we use Rönngren's execution time model (Rönngren, and Ayani 1994) where the simulation execution time T for an observation period when adopting a state saving interval χ is estimated as:

$$T(\chi) = R_{obs} \frac{\delta_s}{\chi} + k_{obs} \delta_r + k_{obs} \delta_c \frac{(\chi - 1)}{2} + R_{obs} \delta_e \quad (3)$$

In this expression, R_{obs} is the number of observed event executions (committed and rollbacked events), δ_s is the average time to save one state of the LP, k_{obs} is the number of observed rollbacks, δ_r is the average time to restore the state of the LP, δ_c is the average execution time for an event in a coast forward phase, and δ_e is the average execution time of an event in normal forward execution. The first term in the sum refers to the estimated time to save states. The last term is the time to execute events during forward execution, and together with the first term it reflects the time to execute the forward execution phase. The second term corresponds to the time to perform rollbacks and restore old states, i.e the rollback phase, and the third term estimates the time spent in coast forward phases, c.p. Equation (2).

The state saving overhead which depend directly on the state saving interval χ is term 2 and 3 in Equation (3):

$$T_{oh}(\chi) = R_{obs}\frac{\delta_s}{\chi} + k_{obs}\delta_c\frac{(\chi-1)}{2} \tag{4}$$

4.3 The ESSS Model

In the ESSS approach we classify each event class i in an LP by the following data and parameter estimations: f_i, the relative frequency of event class i, i.e. the observed number of executed events of class i divided by the total number of observed events; and δ_{ci}, the mean coast forward time for class i events.

Events that belong to an event class with high coast forward times or high granularity are called *heavy* events and correspondingly, events that belong to an event class with low coast forward times or low granularity are called *light* events. In Equation (3) and Equation (4), the execution time is estimated for an χ which does not explicitly account for whether the events in the coast forward phase are light or heavy. To be able to take this difference in event characteristics into consideration when deciding when to save state (provided that events of different classes are evenly distributed in time and events of the same class are not lumped together), we introduce a simulation tuning parameter: p_i, the probability of state saving after events of class i where $0.0 < p_i < 1.0$.

Thus, the average state saving interval χ_{ESSS} for the ESSS model with N event classes can be described as:

$$\chi_{ESSS} = 1/\left(\sum_{i=1}^{N} p_i f_i\right) \tag{5}$$

and the average coast forward event execution time δ_{cESSS} (provided that at least one p_j is less than one, i.e. the simulation state is not saved after each event which would imply copy state saving) as:

$$\delta_{cESSS} = \left(\sum_{i=1}^{N} (1-p_i)\, f_i\, \delta_{ci}\right) / \left(\sum_{j=1}^{N} (1-p_j) f_j\right) \tag{6}$$

Our goal is to reduce δ_{cESSS} in Equation (6) by saving heavy events with higher probability than light events, i.e. for heavy events we would like a high p_i and for light events a low p_i. By this approach it is possible to reduce the re-execution overhead in the coast forward phase.

4.4 Analytical Evaluation of ESSS

To evaluate the ESSS approach we compare the ESSS model to a sparse state saving (SSS) model with a fixed state saving interval given by minimizing $T_{oh}(\chi)$ in Equation (4), see Equation (1). We choose first to analytically study a simple model with the following parameters:

We have two event classes with events from each class equally frequent, i.e. N=2 and f_1=f_2=0.5. Events of class i=2 are considered heavy with a heavy factor HF, i.e. δ_{c2}=$HF\delta_{c1}$. To find an estimate on how well the ESSS model performs we compare the state saving overhead when varying p_1 and p_2. The mean coast forward time for the ESSS model is described by Equation (6) and for the SSS model the mean coast forward time is:

$$\delta_{cSSS} = \sum_{i=1}^{N} f_i\, \delta_{ci} = \frac{\delta_{c1}+\delta_{c2}}{2} = \frac{\delta_{c1}(1+HF)}{2} \tag{7}$$

A comparison can be performed by estimating the simulation state saving overhead during an observation interval for ESSS and SSS respectively using Equation (4):

$$\frac{T_{ohESSS}(\chi)}{T_{ohSSS}(\chi)} = \frac{\left(\frac{\delta_s}{\chi} + \frac{k_{obs}}{R_{obs}}\delta_{cESSS}\frac{(\chi-1)}{2}\right)}{\left(\frac{\delta_s}{\chi} + \frac{k_{obs}}{R_{obs}}\delta_{cSSS}\frac{(\chi-1)}{2}\right)} \tag{8}$$

An important Time Warp simulation measurement is the efficiency parameter E which shows the ratio between the total number of committed events and the total number of processed events (both committed and rolled back events). This parameter together with the average rollback length RL, i.e. the average number of events a straggler causes the simulation to roll back across, can be used to describe the rollback frequency k_{obs} / R_{obs}:

$$\frac{k_{obs}}{R_{obs}} = \frac{1-E}{RL} \tag{9}$$

Furthermore we assume that the time to save the entire LP state δ_s in this simple model is the same as the time to re-execute or coast forward over a light event, i.e. $\delta_{c1} = \delta_s$. The state saving interval χ that minimizes the state saving overhead for SSS can be estimated by Equation (1) with δ_c equal to δ_{cSSS}. This gives the χ_{SSS} estimation:

$$\chi_{SSS} = \left\lceil \sqrt{2\frac{RL}{(1-E)}\frac{\delta_s}{\delta_{cSSS}}} \right\rceil \approx \left\lceil \sqrt{2\frac{RL}{(1-E)}\frac{2}{(1+HF)}} \right\rceil \qquad (10)$$

and the simulation state overhead comparison in Equation (8) can be estimated as:

$$\frac{T_{ohESSS}(\chi)}{T_{ohSSS}(\chi)} = \frac{\frac{\delta_s}{\chi_{ESSS}} + \frac{(1-E)}{RL}\delta_{c_{ESSS}}\frac{(\chi_{ESSS}-1)}{2}}{\frac{\delta_s}{\chi_{SSS}} + \frac{(1-E)}{RL}\delta_{c_{SSS}}\frac{(\chi_{SSS}-1)}{2}} \qquad (11)$$

If we for this analytical model vary the heavy factor *HF* and efficiency *E*, for a fixed rollback length *RL* of 2, we get the results in Figure 2 and 3 respectively. In Figure 2 we can see that the possible relative gain in reduced state saving overhead when using the ESSS state saving model is increased as the heavy factor is increased, i.e. a higher variance in the event coast forward time has a larger impact on state saving performance. For the situation when $\delta_{c2}=8\delta_{c1}$, the optimum choice of the state saving probabilities are p_1=0.0 and p_2=1.0, i.e. states are only saved after the heavy event (i=2). Performance improvement in the overhead directly associated with the state saving for the ESSS model compared to SSS is in this case close to a factor two.

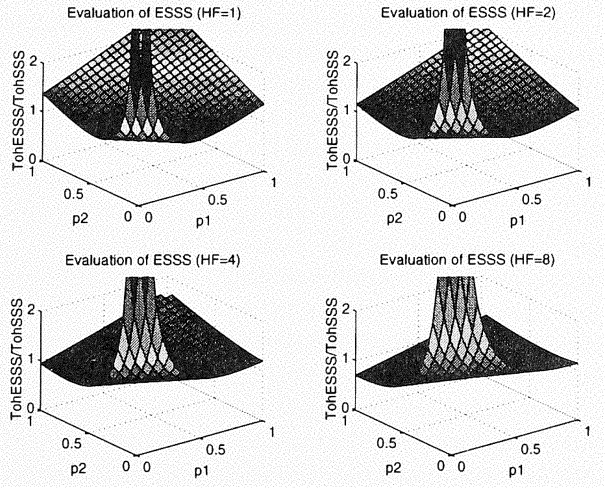

Figure 2: Analytical Estimation of State Saving Overhead for ESSS compared to SSS when varying the Heavy Factor *HF* (E=0.8, f_1=f_2=0.5)

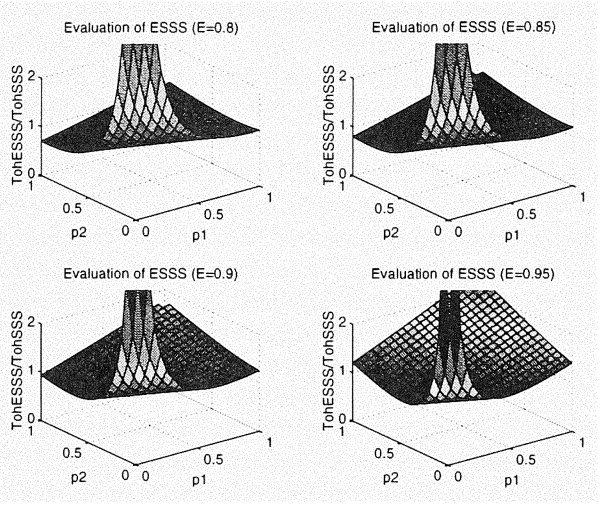

Figure 3: Analytical Estimation of State Saving Overhead for ESSS compared to SSS when varying the Efficiency *E* (*HF*=8, f_1=f_2=0.5)

When we vary the efficiency in the simple analytical example (see Figure 3) we can see that for lower efficiency (around 80%), i.e. for cases when the LPs are performing frequent rollbacks, the performance improvement is larger than for higher efficiencies. This also indicates that our attempt of targeting the coast forward phase could be a promising approach.

Summarizing the results from our analytical evaluation of ESSS, we claim that the ESSS model has a good potential to reduce the state saving overhead in sparse state saving approaches for simulation models which show a large variance in coast forward execution times.

5 PERFORMANCE EVALUATION

The impact of the ESSS model has been evaluated by implementation of the ESSS model on a Time Warp simulator targeted to shared memory multiprocessor workstations. The simulator is written in C and uses aggressive direct cancellation (i.e. anti-messages are sent immediately upon rollback) and static assignment of LPs to processors. GVT calculations as well as fossil (garbage) collection are performed once every second and on demand. Synchronization primitives are supplied by the p4 macro library (Butler, and Lusk 1994). The experiments were performed on a SUN multiprocessor workstation with 4 processors and 128 MB internal memory.

As simulation models we have selected a synthetic model, in which we could modify the parameters and hence the model behavior, as well as a real world simulation model which describes an existing physical system.

5.1 Experiments with a Synthetic Model

The synthetic model is a modification of the PHOLD model (Fujimoto 1989) which is a closed stochastic queueing model with a fully connected net of 64 nodes. Two types of equally probable messages are forwarded within the net. One type of message is labelled *light* and triggers an event that takes 30 microseconds to process, the other is labelled *heavy* and the time to execute the corresponding event is *HF* times longer. The message population is constant and messages are routed with 20% probability to the sending process and with 25% probability to one of eight so called *hot spots* that randomly move around in the network. Otherwise messages are equally likely to be routed to any of the other nodes. This model could be seen as a simple model of a mobile communication system with 64 nodes and 8 mobile terminals moving around in the system. The lighter messages trigger mobile position updates, whereas the heavier messages correspond to resource allocation events which typically are more costly.

Each node's state size is 1Kbyte and the time to save that state is approximately the same as the coast forward time for the lighter event, i.e. $\delta_s/\delta_{clight}=1$. In the experiments we measured the event rate, the efficiency and the rollback length for various combinations of the two state saving probabilities p_1 (light event) and p_2 (heavy event). In each experiment approximately 300,000 events were committed, and each experiment was run three times.

For the first set of experiments, the heavy factor *HF* is eight and in average there is one light and one heavy message per node. The efficiency for this model is around 80%, i.e. this synthetic model is very similar to one of the analytical models studied in Section 4.4.

From the analytical analysis we would expect to get the best performance for the situation when states rarely are saved after light events (p_1 is close to zero) and almost always are saved after heavy events (p_2 is close to one). From the simulation results for the first set of experiments with the above described synthetic model, see Figure 4, we can see how varying the state saving probabilities affect the event rate, i.e. committed events per second. The results show that for this synthetic model the performance is improved when state is rarely saved after light events and more probably after heavy events, as would be expected from the analysis in Section 4.4. However, the best performance is not for $(p_1, p_2) = (0.0, 1.0)$ but for (0.0, 0.6), i.e state should not be saved after every heavy event, but approximately after every second heavy event even though the performance difference is very small compared to $(p_1, p_2) = (0.0, 1.0)$.

If we compare the ESSS performance for the synthetic simulation model and the first set of experiments with the best simulation performance when employing other sparse state saving techniques, we get the results in Table 1. However, these results show that the ESSS model does not outperform the sparse state saving method with fixed state saving interval of 4. This contradicts what we might have expected after studying the analytical analysis in Section 4.4. One factor contributing to this is the high variance in the actual state saving interval for the ESSS model compared to sparse state saving with fixed χ.

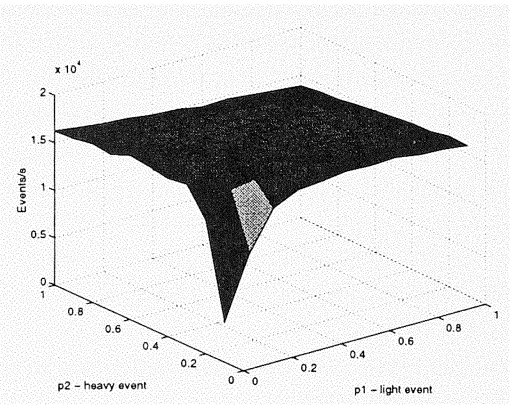

Figure 4: Simulation Results from the PHOLD Model with Hot Spots (f_1=f_2=0.5, *HF*=8)

Table 1: Comparing Simulation Performance for the PHOLD Model with Hot Spots (f_1=f_2=0.5, *HF*=8)

State saving model	Mean χ	Efficiency [%]	Rollback length	Event rate [events/s]
Copy SS	1	79.82	1.85	14,073
Sparse SS	4	85.39	1.75	17,403
Adaptive SS	4.39	84.85	1.76	16,683
ESSS (p_1, p_2)=(0.0, 0.6)	3.32	83.97	1.78	17,223

For this PHOLD simulation model, rollbacks are typically short, only one or two events are rolled back, and even the rather high rollback frequencies of approximately 10% do not introduce high state saving overhead compared to the time needed for normal forward execution. The ESSS model shows better performance than Rönngren's adaptive state saving method, which partly can be explained by the overhead introduced by the adaptability in Rönngren's model. The traditional copy state saving that saves simulation state after each processed event is about 20% slower than the ESSS model as well as the sparse state saving method with χ=4.

For a second set of experiments, the heavy factor *HF* is sixty and in average there are two light and one heavy messages per node, i.e. f_1=2/3 and f_2=1/3. The efficiency for this model is lower than for the first set of experiments, around 75%. The reason to study this synthetical model is its resemblance with the real model to be studied in Section 5.2.

The results from the second set of experiments show that for a model with high variance in event granularity, the best performance is when state is rarely saved after light events and more frequently saved after heavy events. For this model the best performance is for (p_1, p_2) = (0.0, 1.0), see Figure 5.

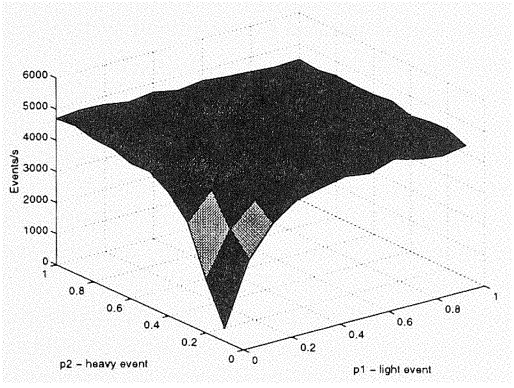

Figure 5: Simulation Results from the PHOLD Model with Hot Spots (f_1=2/3, f_2=1/3, HF=60)

A performance comparison for different state saving methods for the second model is found in Table 2. This PHOLD model also has short rollbacks, about two events in each rollback phase. However, the variance in event granularity is considerably higher for this model which results in increased rollback frequencies as well as decreased optimal state saving intervals. The ESSS model shows better performance than the other state saving methods for this set of experiments. The maximal event rate measured for ESSS is approximately 4% better than the event rate for Rönngren's adaptive method.

Table 2: Comparing Simulation Performance for the PHOLD Model with Hot Spots (f_1=2/3, f_2=1/3, HF=60)

State saving model	Mean χ	Efficiency [%]	Rollback length	Event rate [events/s]
Copy SS	1	74.57	2.09	4,558
Sparse SS	3	78.16	2.03	4,542
Adaptive SS	2.87	78.00	2.09	4,590
ESSS (p_1, p_2)=(0.0, 1.0)	2.49	75.49	2.12	4,772

5.2 Experiments with a Real World Model

The third simulation model is a mobile telecommunication network simulator which models a simple "ring" highway with uniformly distributed base stations. All cells are of equal size and have the same number of channels available. Calls from mobile users arrive according to a Poisson process and have exponentially distributed call durations. An initiated call is placed at a uniformly distributed random position on the highway. Mobility is modeled as a constant truncated Gaussian distributed random speed in one of the two possible directions. Each cell is simulated as an LP. The arrival of calls are generated by a central generator LP, and handover between cells is attempted by sending messages between cell LPs.

The model used for the performance evaluation include 20 base stations, each with 10 available channels, and each in a cell with size 2000 meters. Calls arrive at the rate of one call per second and with an average holding time of 120 seconds. The mobile terminals move with an average speed of 90 kilometers/hour. After an initialization phase, the mobile terminals will be approximately uniformly distributed over the cells.

The two most frequent types of events are *position updates* and *resource allocations* which account for about two thirds and one third of all events respectively. A resource allocation (heavy event) takes approximately 60 times longer than a position update (light event) due to the computation needed to calculate signal to interference ratios when allocating channel resources. For this model, the time to save the state of an LP is approximately the same as the coast forward time for the lighter event, i.e. δ_s/δ_{clight} =1. In the experiments we measured the event rate, i.e. the number of committed events per second for various combinations of the two state saving probabilities p_1 (light event) and p_2 (heavy event). In each experiment 300,000 events were committed, and each experiment was run three times.

Comparing the ESSS performance for this model with the best performance for the other three sparse state saving approaches, we get the results in Table 3:

Table 3: Comparing Simulation Performance for Mobile Communication Model

State saving model	Mean χ	Efficiency [%]	Rollback length	Event rate [events/s]
Copy SS	1	88.38	116.38	9,464
Sparse SS	12	87.46	129.39	10,150
Adaptive SS	12.70	87.48	126.72	10,089
ESSS (p_1, p_2)=(0.05, 0.15)	10.06	87.58	137.11	10,124

These results show that the ESSS model detects the difference in event granularity and hence the best performance for the ESSS model is for a lower mean state saving interval compared to the sparse and adaptive state saving methods. The choice of parameters (p_1, p_2) = (0.05, 0.15) indicates that the ESSS optimal performance is reached when the probability of saving state after light events are lower than after heavy events. This is in accordance with the analytical model. For this simulation model the ESSS, sparse and adaptive state saving methods reach about the same peak event rate. This is somewhat contradictory to the results from the second set of experiments with the synthetic model with high variance

in event granularity. The reason for this is the difference in model geography. The synthetic, fully connected, model has a larger fan-out than the one dimensional real model which results in a higher number of rollbacks and shorter rollback lengths for the synthetic model. This difference indicates that the ESSS model has a larger potential for models with many and short rollbacks. This can be understood by considering the fact that most rollback recoveries start with a coast forward phase. In the case of ESSS there is a lower probability that the first event in a coast forward phase is a heavy event when ESSS is used compared to the other methods for sparse state saving. Thus for a constant efficiency ESSS is favored by a higher number of short rollbacks as compared to cases with fewer and longer rollbacks.

6 CONCLUSIONS

In this paper we have introduced a novel approach for sparse state saving in Time Warp synchronized discrete event simulation, event sensitive state saving (ESSS). This method is based on the observation that the time to execute the coast forward phase in the simulation progress cycle is dependent on the re-execution time of the coast forwarded events. For many important types of simulation models such as battle field simulations and simulations of mobile communication systems there is a significant difference in the coast forward cost for different types of events. This has not been captured in previously proposed methods for sparse state saving. In contrast, the ESSS state saving method uses information on the coast forward cost for individual events to decide when to save or checkpoint a state of logical processes.

In this paper we have presented an analytical analysis of the potential benefits from using an event sensitive approach. This analysis indicates that overhead due to sparse state saving in some cases can be reduced by nearly 50% when using the ESSS approach compared to other methods presented in the literature. Results for synthetic as well as realistic simulation models show that the ESSS model is a promising approach for simulation models where event granularity or execution time has large variance. ESSS is in particular favored by short rollbacks which often occurs in real well behaved simulation models. Our experimental results show that the ESSS approach could result in a more than 4% over all performance improvement to other state saving methods.

The result from this work shows that an efficient state saving method which takes more of the simulation modelling details (such as event granularity) into consideration is likely to improve state saving performance. We plan to continue this work and implement adaptive versions of the proposed method for sparse state saving.

REFERENCES

H. Bauer, and C. Sporrer. 1993. Reducing Rollback Overhead in Time Warp Based Distributed Simulation with Optimized Incremental State Saving. In *Proc. of the 26th Annual Simulation Symposium*, 12-20.

R. Butler, and E. Lusk. 1994. Monitors, messages, and clusters: the p4 parallel programming system. *Parallel Computing*: 20.

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

R. M. Fujimoto. 1989. Time Warp on a Shared Memory Multiprocessor. *Transactions of the Society for Computer Simulation*, Vol. 6, No. 3:211-239.

R. M. Fujimoto. 1990. Parallel Discrete Event Simulation. *Comm. of the ACM*, Vol. 33, No. 10:30-53.

D. R. Jefferson. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3:404-425.

Y.-B. Lin, B. R. Preiss, W. M. Loucks, and E. D. Lasowska. 1993. Selecting the Checkpoint Interval in Time Warp Simulation. In *Proc. of the 7th Workshop on Parallel and Distributed Simulation*, 3-10.

A. C. Palaniswamy, and P. A. Wilsey. 1993. An Analytical Comparison of Periodic Checkpointing and Incremental State Saving. In *Proc. of the 7th Workshop on Parallel and Distributed Simulation*, 127-134.

B. R. Preiss, W. M. Loucks, and I. D. MacIntyre. 1994. Effects on the Checkpoint Interval on Time and Space in Time Warp. *ACM Transactions on Modeling and Computer Simulation*, Vol. 4, No. 3:223-253.

R. Rönngren, and R. Ayani. 1994. Adaptive Checkpointing in Time Warp. In *Proc. of the 8th Workshop on Parallel and Distributed Simulation*, 110-117.

R. Rönngren, M. Liljenstam, J. Montagnat, and R. Ayani. 1996. A comparative study of state saving mechanisms for Time Warp synchronized parallel discrete event simulation. In *Proc. of the 29th Annual Simulation Symposium*, 5-14.

AUTHOR BIOGRAPHIES

SVEN SKÖLD is a Ph.D. student in the Department of Teleinformatics at the Royal Institute of Technology, Stockholm, Sweden. His research interests include efficient simulation techniques and in particular parallel discrete event simulation.

ROBERT RÖNNGREN is an associate professor in the Department of Teleinformatics at the Royal Institute of Technology, Stockholm, Sweden. His research interests include sequential and parallel discrete event simulation.