Effects of the Checkpoint Interval
on Time and Space in Time Warp*†

Bruno R. Preiss
Wayne M. Loucks
Ian D. MacIntyre‡

Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1

“This work was supported in part by the Information Technology Research Centre (ITRC) of the Province of Ontario (Canada) and by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

†An earlier version of a part of this paper appeared in the 1992 SCS/IEEE Workshop on Parallel and Distributed Simulation under the title “On the Trade-off between Time and Space in Optimistic Parallel Discrete-Event Simulation” [27].

‡Ian MacIntyre is currently at CARP Systems International, 600 Terry Fox Drive, Kanata, Ontario, Canada, K2L 4B6.
Abstract

Optimistically synchronized parallel discrete-event simulation is based on the use of communicating sequential processes. Optimistic synchronization means that the processes proceed under the assumption that a synchronized execution schedule is fortuitous. Periodic checkpointing of the state of a process allows the process to roll back to an earlier state when synchronization errors are detected. This paper examines the effects of varying the checkpoint interval on the execution time and memory space needed to perform a parallel simulation.

The empirical results presented in this paper were obtained from the simulation of closed stochastic queueing networks with several different topologies. Various intra-processor process scheduling algorithms and both lazy and aggressive cancellation strategies are considered. The empirical results are compared with analytical formulae predicting time-optimal checkpoint intervals. Two modes of operation, throttling and thrashing have been noted and their effect examined. As the checkpoint interval is increased from one, there is a throttling effect among processes on the same processor which improves performance. When the checkpoint interval is made too large, there is a thrashing effect caused by interaction between processes on different processors. It is shown that the time-optimal and space-optimal checkpoint intervals are not the same. Furthermore, a checkpoint interval that is too small adversely affects space more than time; whereas, a checkpoint interval that is too large adversely affects time more than space.

Categories and Subject Descriptors: C.4 [Performance of Systems]—Modeling

General Terms: Performance, Theory

Additional Key Words and Phrases: checkpointing, discrete event simulation, parallel simulation, Time Warp

1 Introduction

This paper examines the trade-off between execution time and memory space in optimistically synchronized parallel discrete event simulation. Optimistic synchronization assumes that the concurrent execution of simulation processes is naturally synchronized. When a process detects the loss of synchronization, it returns to an earlier state (presumably, one in which synchronization has not been lost) and resumes execution.

In order to use optimistic synchronization, processes must be able to return to earlier states. This is accomplished by periodically checkpointing the state of the process and by reconstructing the desired state from a checkpointed one. Hence, there is a time penalty associated with the reconstruction of a state and both time and space penalties associated with the checkpointing of a state.

The next section introduces parallel discrete event simulation and describes an implementation of the most common optimistic synchronization method. It also covers process
scheduling, message cancellation, and introduces the checkpoint interval. Section 3 describes the experiments performed to assess the space/time trade-off. The experimental results are presented in Sections 4, 5, and 6. Section 4 examines the effects of the checkpoint interval on total execution time. It also examines the impact of thrashing and throttling on the performance of the simulation. Section 5 shows the effects of the checkpoint interval on the maximum memory requirements of a simulation. Section 6 shows the trade-off between space and time. Finally, we summarize our observations in Section 7.

2 Parallel Discrete-Event Simulation

In parallel discrete-event simulation, interacting physical processes are represented in the simulation by concurrent communicating logical processes (LPs). Each LP has its own notion of simulation time (local virtual time or LVT) and the LPs exchange timestamped messages[17].

Optimistic synchronization allows each LP to execute asynchronously. However, to ensure correct simulation results, certain causality constraints must be met. Specifically, each LP must process received messages in non-decreasing timestamp order.

2.1 Optimistic Synchronization

The purpose of synchronization in parallel discrete-event simulation is to ensure that each LP processes its input messages in non-decreasing timestamp order. Synchronization methods
fall into two broad categories—conservative and optimistic[28].

In conservative methods, execution of an LP is halted until it can be determined with certainty that resumed execution will not violate causality. Conservative methods are prone to deadlock[4].

In optimistic methods, LPs always process input messages whenever they are available under the implicit assumption that the message sequence does not violate causality. When a causality violation occurs, the LP is returned to its state immediately prior to the violation (rollback), and execution resumes[11].

In this paper we focus on the most common optimistic synchronization method, namely Time-Warp[10]. Time-Warp facilitates optimistic synchronization in the following manner: Each LP has an input queue of unprocessed messages (the future queue) in which all of the messages have timestamps greater than or equal to the LVT, and a queue of processed input messages (the past queue) in which all of the messages have timestamps less than or equal to the LVT. As input messages are processed, they are moved from the future queue to the past queue. Each LP also maintains a queue of past states with timestamps less than the LVT.

As an LP executes, it sends output messages. When an LP rolls back, it may have to cancel the effects of some or all of the output messages. This is accomplished by sending an antimessage for each output message that must be cancelled. When an antimessage encounters its partner, the pair of messages annihilate, i.e., they are discarded. Each LP maintains an output queue of antimessages. Every time an output message is sent, its partner
antimessage is inserted into the output queue.

When an LP receives a message with a timestamp less than its LVT (a straggler) the LP must perform a rollback. The LP restores an earlier state from its state queue, and moves processed messages from the past queue back into the future queue. If the straggler is an antimessage, it will annihilate with a message in the future queue. Otherwise, the straggler is inserted into the future queue.

2.2 Cancellation Strategies

When a rollback occurs, some output messages may need to be cancelled. There are two categories of cancellation strategy—aggressive and lazy.

Aggressive Cancellation When the aggressive cancellation strategy is used, antimessages are sent immediately when a rollback occurs. Such messages often lead to secondary rollbacks in other LPs. The assumption is that the cancelled messages may be causing erroneous computation in other LPs.

Lazy Cancellation When the lazy cancellation strategy is used, antimessages are not sent when a rollback occurs. Instead, antimessages are placed into a queue of pending antimessages. When the LP resumes execution, it will generate output messages. In the event that an output message is the same as a message that would have been cancelled during the rollback, then the new output message and its antimessage will be discarded[7].
The assumption is that after a rollback, an LP is likely to produce the same output messages. In this case, the lazy cancellation strategy will reduce the unnecessary secondary rollbacks that would occur with aggressive cancellation.

2.3 State Saving Costs

In order to permit rollback, each LP must periodically save (all or part of its) state (referred to as checkpointing). Checkpointing has two adverse effects on the simulation: First, memory is consumed and it is possible that there will be insufficient memory to permit the simulation to complete in any amount of time. Second, processing time is needed to copy the portion of the state to checkpointed. There are several ways that the overhead (both time and space) can be addressed in an optimistic simulation.

2.4 Checkpoint Interval

One approach to reducing the state-saving overhead is to increase the checkpoint interval. This reduces the memory requirements by reducing the number of states which must be preserved and reduces the number of CPU cycles used to copy those states. Unfortunately, there are several issues which counteract some of these gains. In the simplest case, each LP checkpoints its state before processing each input message. In the more general case, each LP checkpoints its state before processing every $n^{th}$ input message, where $n \geq 1$. The checkpoint interval, $I_{cp}$, is the number of input messages processed between two checkpoint
operations. Thus, $I_{cp} = 1$ corresponds the simple case of checkpointing before each input message$^1$.

If a rollback occurs when $I_{cp} = 1$, then the required state of the LP (i.e., its state immediately prior to the timestamp of the straggler) can be restored directly from the state queue. This is because the state queue contains every possible state to which a rollback may be required. On the other hand if a rollback occurs when $I_{cp} > 1$, the required state may not be in the state queue. In this case, the LP must roll back to an earlier checkpointed state and recompute the required state from the earlier state. The required state is recomputed by reprocessing input messages. Note that the time penalty associated with $I_{cp} > 1$ is the time required to reconstruct the uncheckpointed states.

An important consideration when implementing Time-Warp with $I_{cp} > 1$ is how to reconstruct the uncheckpointed state when it is needed. The state is easily reconstructed by reprocessing input messages. However, output messages regenerated during state reconstruction must not be sent since they are an artifact of the state reconstruction process. An LP that is reconstructing a missing state is said to be coasting forward$^6$. The reconstruction

---

$^1$Actually, in our implementation, all input messages having exactly the same timestamp are processed by an LP at once. A set of input messages having the same timestamp is called an input event combination. For a discussion on the importance of using event combinations, see [16]. For the sake of clarity, in this paper we refer simply to input messages. We implicitly assume that should there be simultaneous input messages, all of these are processed by an LP at once. In fact, for the benchmarks presented in this paper, an overwhelming majority of event combinations consist of exactly one input message.
time is a function of the time needed for one event to be reprocessed \((T_{event})\), the number of events to be reprocessed, and the probability that a state must be reconstructed.

2.4.1 Checkpoint Processing Time

The *checkpoint time*, \(T_{cp}\), is the time required to perform a checkpoint operation. It consists of two parts—the time required to allocate a state buffer, and the time required to copy the current state into the state buffer. This time \((T_{cp})\) is a direct addition to the processing time of every LP activation which requires its state to be checkpointed.

An alternative to checkpointing on the basis of simulation events is to checkpoint on the basis of real CPU execution time. For example, in the original design of the Time Warp Operating System\cite{1}, the state of an LP was checkpointed after that LP had consumed a *save period* amount of real CPU execution time. However, early tests showed that for unsaturated workloads using a save period of zero always resulted in the best execution times\cite{1}.

The effect of \(T_{cp}\) on the execution time of the simulation is not the same in all cases. In addition to the impact of the relative sizes of the checkpointing time and the event processing times as discussed in the rest of the paper, there is a question as to the impact of CPU load on the importance of the checkpoint processing time. In fact, if a processor has no *productive* simulation work to be performed, then the cost (in CPU time) of checkpointing can be ignored. This is demonstrated in the work described by Bellenot\cite{1} with regards to *saturated* and *unsaturated* simulation execution.

In the empirical study described starting in Section 3, we have examined simulation cases
where the workload is sufficiently high to cause the time expended on checkpointing to have an impact on the simulation execution time. This is what Bellenot calls *saturated* simulation.

### 2.4.2 Global Virtual Time Estimation Overhead

The global virtual time (GVT) of an optimistically synchronized simulation is the minimum of the LVTs of all LPs and of the timestamps of any messages in transit. The GVT is an important concept in optimistic parallel simulation as it indicates the progress of the simulation and is used to facilitate the recovery of storage. If the checkpoint interval is one, no LP will ever roll back to a time before the GVT. In this case, checkpointed states with LVT values and messages with timestamps less than GVT are no longer required (with the exception of the last state).

If the checkpoint interval is greater than one, then it may be that an LP has not checkpointed the state corresponding to GVT. In this case, rather than preserving a state with a virtual time of GVT, the *surrogate-GVT-state* (i.e., the state with the largest virtual time less than or equal to GVT) is preserved. The surrogate-GVT-state and any messages that arrived with a timestamp value between its virtual time and GVT are used to reconstruct the missing state, should it become necessary for the LP to roll back to GVT. As long as the simulation does not need to rollback to GVT then the only memory cost associated with a checkpoint interval greater than one is the extra memory needed to store incoming messages with timestamp values between the virtual time of the surrogate-GVT-state and GVT. Checkpointed states and messages with timestamps less than the virtual time of the
surrogate-GVT-state are no longer required. Items no longer required are called fossils. The process of recovering storage allocated to fossils is called fossil collection.

GVT introduces two further time penalties, the time to estimate GVT and the time perform the fossil collection algorithm. All the experiments described in this paper use the token-based GVT algorithm described in [20].

2.5 Process Scheduling Algorithms

In general, the number of LPs required for a simulation will not match the number of processors available to do the simulation. In particular when simulating large systems, it is likely that the number of LPs will exceed the number of processors. Consequently, each processor is responsible for the execution of more than one LP. In the event that more than one LP is ready to execute, a scheduling algorithm is invoked to select the next LP for execution. As indicated above in Section 2.4.1, the benchmarks used in this simulation produce saturated workloads. I.e., processors rarely have idle time. In this paper we examine several different scheduling algorithms. For a more complete discussion of scheduling algorithms see [2] and [12].

In this paper we only consider non-preemptive scheduling algorithms. I.e., once an LP begins to process an input message, the processing of that message will proceed to completion before another LP (on the same processor) is permitted to execute. Furthermore, the arrival of an antimeessage at an LP will not terminate the current processing cycle of that LP.
2.5.1 Smallest-Timestamp-First Scheduling

Smallest-timestamp-first (STF) scheduling is the most commonly used scheduling heuristic in optimistic simulation systems[12]. The STF scheduling algorithm gives higher priority to LPs having input messages with lower timestamps. This algorithm always chooses for execution the LP from among those having an input message to be processed the one having the message with the smallest timestamp. Use of the STF scheduling algorithm on a single processor is equivalent to the conventional sequential discrete-event simulation method.

Scheduling by smallest message timestamp tends to reduce the number of rollbacks for the experiments discussed in Section 3. This is because the LPs having input messages with lower timestamps are less likely to have to rollback their computation, since the simulation they perform is less “speculative” than LPs that have messages much further ahead (in simulation time).

2.5.2 Smallest-Virtual-Time-First Scheduling

Although the STF scheduling algorithm is optimal on a uniprocessor (i.e., it causes no rollbacks), on a multiprocessor it is only a heuristic. As an alternative, we propose an algorithm in which the scheduling decision is based on the local virtual times of the LPs on the processor where the LVT is taken to be the timestamp of the last message processed by the LP. In order to ensure the steady progress of the simulation, it seems intuitive to give higher priority to LPs with lower LV Ts. The smallest-virtual-time-first (SVTF) scheduling algorithm always chooses the (ready) LP with the smallest LVT. The chosen LP is allowed
to process one input message.

The SVTF scheduling algorithm attempts to advance the GVT by advancing the smallest LVT on each processor. In doing so, it may help the fossil collection process to lessen the memory requirements of the simulation.

2.5.3 Round-Robin Scheduling

In order to show the importance of an appropriate scheduling algorithm and to determine the interaction between checkpointing and scheduling, we introduce the round-robin (RR) scheduling algorithm. The RR algorithm is an oblivious algorithm—it does not make its scheduling decisions on the basis of the simulation times of the LPs or messages. When using the RR scheduling algorithm, LPs that are ready to execute share the processor in round-robin fashion. An LP is ready to execute as long as it has input messages to process. During its turn, an LP processes one event message. When an LP runs out of messages to process, it is blocked. When a blocked LP receives an input message, the LP is placed at the end of the list of ready LPs.

3 Experimental Overview

The results presented in this paper were obtained from parallel simulations of closed stochastic queueing network benchmarks based on those described in [18].

The following is a very brief description of the benchmarks. For more detailed descriptions see [15, 22, 23, 24]. These simulations were coded and run on a Transputer system
(Section 3.3), and a number of experiments were performed on the system (Section 3.4).

3.1 Benchmarks

The system simulated is a static network of nodes. The network is populated by a fixed number of customers. A customer arrives at a node; waits for service; is serviced; and then, departs for another node. Each node has $f_{in}$ inputs, $f_{out}$ outputs, a single queue, and a single server. The queueing discipline is FCFS. Service is non-preemptive. The service time distribution is a biased, exponentially distributed random variable. The smallest service time is $\mu_{min}$; the mean service time is $\mu = 5\mu_{min}$; and, the system was simulated for $1000\mu$ time steps.

The suite of benchmarks includes four network topologies—ring ($f_{in} = f_{out} = 2$), multi-stage ($f_{in} = f_{out} = 2$), mesh ($f_{in} = f_{out} = 4$), hypercube ($f_{in} = f_{out} = 6$). In all cases, the system consists of 64 nodes.

The time-averaged number of customers in each node in the system is a measure of the simulation "load". Simulations were done with loads of one, four, and eight customers per node.

3.2 Logical Process Characteristics

For all of the simulations reported in this paper, the size of the state of each LP was 664 bytes. The state includes: a queue containing the arrival times of the customers awaiting service
(length = 128 integers plus 4 integers to manage the queue), 4 statistics (6 integers and 1
double each), and 1 random number state (2 integers) for a total of 664 bytes per state.
The average event processing time, $T_{\text{event}}$, varied with the load (i.e., the average number of
customers at each node). For a load of one, $T_{\text{event}}$ was 2.2 ms, for a load of four, $T_{\text{event}}$ was
2.9 ms, and for a load of eight it was 3.5 ms.

3.3 Simulation Software and Hardware

The simulations were implemented using the Yaddes distributed discrete event simulation
language[20, 21, 26]. The simulations were performed on a Transputer multiprocessor with a
total of eight processors. The connections between the processors form a cube. In all cases,
the obvious LP to processor assignment was made. Note that, when using eight processors,
each processor was responsible for running eight LPs.

3.4 Experiments

A total of 2592 simulations were run. As the base simulation, we selected the hypercube
topology, an average of four customers per node (i.e., load=4), using eight processors. We
then independently varied the parameters topology, load, and number of processors used.
Table 1 lists the other test cases examined. In group I, the network topology is varied (the
number of nodes in the network is 64 for all topologies); in group II, the load is varied; and
in group III, the number of processors used to perform the simulation is varied.
Each of the test cases listed in Table 1 was run 324 times. Checkpoint intervals of 1, 2, 3, 4, 5, 6, 8, 10, and 15 were used. Both lazy and aggressive cancellation strategies were used. Three process scheduling algorithms were used—STF, SVTF, and RR. In addition, in order to evaluate the impact of increasing the checkpoint time, we introduced a variable delay loop into the checkpoint routine. The effect of this loop is to increase the checkpoint time by a fixed amount called the loop delay. Loop delays of 0 ms, 0.128 ms, 0.256 ms, 0.512 ms, 1.024 ms, and 2.048 ms were used. By comparison, the inherent checkpoint time for the base case simulation is 0.08 ms.

4 Execution Time Results

In this section we present and discuss the results of the benchmark simulations. The emphasis is on the effect of the checkpoint interval on the execution time of the simulation. The maximum storage required to perform the simulation is examined in Section 5.

4.1 Execution Time vs. Checkpoint Interval

Figure 1 shows execution time vs. checkpoint interval for the hypercube topology simulations using eight processors. In this case, the simulations were performed using the inherent checkpoint time—a loop delay of zero. This figure clearly shows that the degree to which the checkpoint interval affects the execution time depends on the process scheduling algorithm used. The best performance is achieved using the STF scheduling algorithm. When using
the SVTF scheduling algorithm the simulation takes between 5 and 30 percent longer than when using the STF scheduling algorithm. The simulation is significantly slower when using the RR algorithm than when using the other two scheduling algorithms.

Figure 1 shows that when using the RR scheduling algorithm, execution time increases as the checkpoint interval is increased. The time-optimal checkpoint interval in this case is one. On the other hand, execution time first decreases, reaches a minimum, and then increases as the checkpoint interval is increased for both the SVTF and STF scheduling algorithms. (The time-optimal checkpoint intervals are tabulated Section 4.3.)

As Figure 1 shows, the execution time vs. checkpoint interval curves for the SVTF and STF scheduling algorithms are flat. I.e., the effect on execution time of using a larger checkpoint interval is minimal. However, we can expect that when using larger checkpoint intervals, the simulation will require significantly less space, since the amount of checkpointed state information will be substantially reduced (see Section 5).

Finally, Figure 1 shows the effect of using lazy vs. aggressive cancellation is negligible for the SVTF and STF scheduling algorithms. On the other hand, aggressive cancellation performs substantially better than lazy cancellation when using the RR scheduling algorithm with checkpoint interval greater than two. Why aggressive cancellation is better than lazy in this case can be explained as follows: Since the RR scheduling algorithm does not consider the simulation times of either messages or LPs, it is more likely that the LVTs of the LPs will not be close together. E.g., some LPs may have LVTs much further ahead than others. The skew in LVTs will result in more erroneous computation, and thus, time wasted. When
aggressive cancellation is used, the LVT skew is reduced because an LP cannot execute very far ahead without being rolled back. On the other hand, when using lazy cancellation, rollbacks are delayed. Consequently, more erroneous computation occurs.

4.2 Effects of Checkpoint Time

Figure 2 shows execution time vs. checkpoint interval for the hypercube topology simulations using eight processors. In all cases, the STF scheduling algorithm and the lazy cancellation strategy were used as these generally result in the best execution times. Figure 2 shows the effects of increasing the checkpoint time. Note, the checkpoint time in these simulations is the inherent checkpoint time required to checkpoint the state of an LP, plus the delay introduced by an artificial delay loop. The delay introduced by the delay loop was varied between 0 ms and 2 ms. The inherent checkpoint time (i.e., for a loop delay of 0) is 0.08 ms and the average event processing time is 2.9 ms.

Figure 2 shows the obvious result that increasing the checkpoint time increases the total execution time. However, a more important observation is that as the checkpoint time is increased, the time-optimal checkpoint interval also increases. In other words, as the time cost to perform a checkpoint increases, then the time-optimal checkpoint interval increases.

Figure 2 also shows that when larger checkpoint costs are incurred (i.e., a larger checkpointing delay) the execution time increases more quickly for checkpoint intervals that are smaller than the optimum as compared with checkpoint intervals greater than the optimum. This confirms the observation originally made by Lin on the basis of simulations of parallel
simulation[13], that when choosing a checkpoint interval, it is better to select an interval that is too large, rather than one that is too small. I.e., err in the direction of checkpointing less often.

4.3 Time-Optimal Checkpoint Intervals

Tables 2 and 3 show the experimentally determined time-optimal checkpoint intervals for the eight simulation test cases listed in Table 1. Results are tabulated for two values of loop delay, for both cancellation strategies, and for the STF and SVTF scheduling algorithms. These tables were constructed by running the simulations using checkpoint intervals of 1, 2, 3, 4, 5, 6, 8, 10, and 15, and tabulating the interval which resulted in the smallest execution time. Also shown in the tables is the execution time for each of the time-optimal checkpoint interval simulations.

Overview The impact of changing the checkpoint interval can be summarized by considering the operation of a rollback in a simulation using aggressive cancellation as shown in Figure 3. Consider a processor that is responsible for the execution of eight LPs, \(LP_1\), \(LP_2\), \(\ldots\), \(LP_8\). Assume that the process rolled back is \(LP_1\). As the checkpoint interval increases, the number of intermediate states which must be reconstructed before the required state is obtained also increases. Furthermore, as the checkpoint interval increases, an LP that must reconstruct a state begins coasting forward from an earlier LVT. In our implementation, the coast-forward operation is atomic. Thus, if the scheduling algorithm selects \(LP_1\) for
execution, the processor is prevented from executing events in the other LPs until the coast-
forward operation is finished. This effectively increases the granularity (i.e., the execution
time) of each rollback.

The increase in granularity has both beneficial and detrimental effects. The increase in
granularity causes $LP_2$ through $LP_5$ to stop execution (and thus to stop the progress of their
LVTs). This in turn permits more of the messages destined for these LPs to arrive before any
one of them acts. Thus, any messages to be received or annihilated can be processed before
the next action of the LP has taken place rather than after another rollback. Additionally,
the scheduler can make a better choice on which LP to activate following completion of the
rollback (if STF scheduling is used). As a result, the increase in the checkpoint interval tends
to defer action until a more correct action can be taken and, thus, decreases the number of
rollback cycles. We call this mechanism *throttling* (in contrast to *thrashing* which is defined
below).

As the granularity increases, the differences between the LVTs of local processes and
the LVTs of remote processes also increase (assuming that the remote processor is not si-
multaneously rolling back). As these differences increase, there is an increasing probability
that some of the remote processes will receive a message (or an antimessage) from either
the LP that is rolling back or from one of the other LPs after the rollback operation has
been completed. This can lead to a *thrashing* condition and a significant increase in the
number of rollback operations (and, thus, a decrease in performance). The major cause of
thrashing is the disparity of event execution times as the checkpoint interval is increased.
Consider the following example: In the base case, the event execution time is 2.9 ms and the state saving time is 0.08 ms. When the checkpoint interval is 15, which means that 7.5 states must be reconstructed on average, a rollback/coast forward sequence requires 21.75 ms (7.5 × 2.9 ms) before antimessages or real messages can be processed by the other LPs at the same processor. This delay also represents the delay before any message can be sent to LPs in other processors. During this time, the LPs in other processors might compute seven or eight new (potentially erroneous) states. Thus, each rollback increases the potential number of secondary rollbacks as the checkpoint interval is increased.

Figure 4 shows the effect of *throttling* and *thrashing* on the performance of the simulation of four different topologies. The differences among the topologies are discussed below. Figure 4(a) indicates that for small values of checkpoint interval the number of rollback cycles decreases as the checkpoint interval increases (*throttling*). At some point this trend reverses as *thrashing* starts. It is of note that there is a significant change in the number of rollback cycles as the checkpoint interval is varied. This phenomenon is one of the sources of error in the theoretical results described in Section 4.4.

Figure 4(b) verifies that as the checkpoint interval increases, despite fewer rollback cycles the number of events evaluated remains approximately the same (the number needed for the (uniprocessor) simulation is between 52,000 and 54,500 depending on the topology). The fact that the number of events evaluated does not change indicates that while the number of antimessages does not change, fewer of them cause a rollback cycle (as discussed previously). Figure 4(c) shows this information in a different form, which clarifies the relationship between
the two phases of operation. During throttling there is an increase in the length of the rollback cycle as the checkpoint interval is increased, while thrashing causes the rollback cycle length to decrease as the checkpoint interval is increased.

**Effects of Topology** The simulation results used to determine the Group I values shown in Table 2 have been plotted in Figure 5 (for lazy cancellation). This figure shows that (for all but the ring topology) as the checkpoint interval is varied from 1 to 15, the number of rollbacks first decreases and then increases beyond a critical value of checkpoint interval. Furthermore, minor fluctuations in the execution time occur for checkpoint interval values below a critical value. This indicates the two phases of operation—throttling, which occurs as the checkpoint interval is increased but remains under a critical value, and thrashing, which occurs as the checkpoint interval is increased beyond the critical value.

The general performance of the various topologies, as indicated in Figure 5 is very similar with the exception of the ring topology. The ring is a very special case as a result of the way in which the LPs are allocated to the processors. Since the ring has eight consecutive nodes in one processor, there are only four (unidirectional) links between LPs on one processor and LPs on other processors. Table 4 lists the number of links between LPs on the same processor and the number of links external to each processor.

As the fanout of a node increases, the potential to impact a remote LP negatively also increases. Thus, the potential for thrashing increases. As a result, the time-optimal checkpoint interval for the various topologies tends to increase as the fanout decreases (see Table 2).
The multistage network is the one exception. This is because the multistage network is exceptional in its structure. Unlike the other topologies in which nodes are interconnected with bidirectional links, the multistage network receives messages from one set of nodes and issues messages to a completely different set of nodes. It is clear from Table 4 that the multistage topology more closely resembles the mesh (counting external links) or the hypercube (counting adjacent processors) than the ring (counting links per node). Thus, although the fanout appears to have an impact within similar structures, the fanout is not the only controlling factor.

Effects of Load  The load is a measure of the average number of customers in each LP. The average number of customers in each LP affects the repeatability of the behaviour of LPs during forward simulation after a rollback. Since the customers are indistinguishable, the behaviour of an LP experiencing rollback is “repeatable” in the following sense: After a customer arrives at an LP with a busy server and causes that LP to roll back, that customer joins the end of the queue. However, since the server is busy, it does not really matter to an external observer where in the queue the customer is placed because customers are indistinguishable. As the LP resumes forward simulation after the rollback, its external behaviour (observed as customer departures) will be the same as the behaviour prior to the rollback. As long as the server is busy and there are customers in the queue, the LP’s external behaviour is repeatable. Therefore, the performance when using lazy cancellation is expected to improve as the load increases. Results for simulations with loads of 1, 4, and 8
are given in Figure 6 and the time-optimal checkpoint interval values are listed in Table 2. It is clear from the figure that the smaller-load simulations enter the thrashing condition for much smaller checkpoint intervals. This is also reflected in the time-optimal checkpoint interval values. In most cases the table shows the expected result that for smaller load, the optimal checkpoint interval tends to be smaller, since the behaviour of the LPs after rollback is less repeatable. On the other hand for larger loads, larger checkpoint intervals can be used. These results also suggest that an unsaturated workload is likely to cause thrashing for smaller checkpoint intervals than a saturated workload.

**Effects of Number of Processors** The performance of the simulation with 2, 4, and 8 processors is shown in Figure 7 and in Table 2. The results indicate that thrashing occurs with the larger number of processors and the largest checkpoint interval. This is caused by an increase in the number of processors that depend on the results of any process which must rollback. The time-optimal checkpoint intervals, shown in the Table 2, indicate that as the number of processors is decreased, the optimum checkpoint interval increases. This sort of behaviour is expected because, in the limit when run on a sequential processor, the STF scheduling algorithm executes the various LPs in the same order and at the same values of LVT as they would be executed in a sequential execution of the same problem. Since a sequential simulation never rolls back, the optimum checkpoint interval is infinite. As the number of processors is increased, STF scheduling behaves less like a sequential simulation and the optimum checkpoint interval decreases. These effects are also consistent with those
described by Bellenot[1].

Effects of Scheduling Strategies  The performance for the different strategies are shown in Figure 8. Tables 2 and 3 give the time-optimal values for the STF and SVTF scheduling algorithms. The performance results indicate that the RR scheduling algorithm is usually in the thrashing state and that any increase in the cost of rolling back seriously affects the overall system performance. The performance of STF scheduling is fairly constant, regardless of the checkpoint interval, while the SVTF scheduling algorithm starts to thrash for fairly modest sizes of checkpoint interval. Although some of the trends in the time-optimal checkpoint interval discussed above are noticeable in the RR and SVTF scheduling strategies, there are also many inconsistencies. These inconsistencies result from the poor performance of these algorithms at large checkpoint interval values.

Effects of Checkpointing Time  The impact of increasing the checkpoint processing time on the validity of the previous descriptions can be examined by comparing the trends in Figure 9 and the time-optimal values given in Table 2. In most cases the results for both the intrinsic delay and the added 1 ms delay for checkpoint execution produce the same trends. It is clear from Figure 9 that as the cost of checkpointing increases, the execution times for the simulation runs with the smaller checkpoint intervals also increase (and the time-optimal checkpoint interval also increases). This is due to the fact that a significant amount of time is being used in checkpointing many states.
Summary  The trends described above are based on this particular empirical study only. However an understanding of these trends should be advantageous to determining situations where altering the checkpoint interval may be beneficial from execution time considerations. These results show that the execution time of simulation problems with some or all of the following characteristics could benefit from the use of checkpoint interval values larger than one:

- systems which exhibit few secondary rollbacks,

- systems which exhibit few changes as a result of rollbacks, and

- systems which will experience a saturated parallel simulation execution.

We have also conducted experiments in which we varied the lookahead of the node implementation—the lookahead is a measure of the node's predictiveness. (For a detailed discussion of lookahead see [5].) These results, which are reported in [25], suggest that systems which exploit a high degree of predictiveness to limit the negative impact of aggressive cancellation can also benefit from the use of checkpoint intervals larger than one.

4.4 Comparison of Theoretical Bounds and Experimental Results

Selection of the optimal checkpoint interval has been extensively studied in the context of transaction-oriented database systems[3, 8, 9, 19, 29, 30]. In the context of transaction systems, the purpose of checkpointing and rollback recovery is to improve the reliability of the
system in the presence of failures. In optimistic simulation, the mechanism of checkpointing and rollback recovery is used for synchronization. Because of this, the arrival processes for rollback occurrences are fundamentally different and the results from transaction-oriented systems are not directly applicable to simulation. More recently, Lin and Lazowska have studied the selection of time-optimal checkpoint intervals specifically in the context of optimistically synchronized parallel simulation[13]. In this section we compare the experimentally determined time-optimal checkpoint intervals with the Lin-Lazowska interval and with the theoretical upper and lower bounds[14].

4.4.1 Time-Optimal Checkpoint Intervals

In [13], Lin and Lazowska first derived bounds for the state saving time overhead. Using these bounds, it is possible to select a checkpoint interval which minimizes the state-saving time overhead. In this paper we will briefly present the results of their derivation. We refer the interested reader to [13] and [14] for the details of the derivation. To present their results, we first introduce some notation:

$I_{cp}$ **checkpoint interval**: The number of input messages processed between two consecutive checkpoint operations.

$N_{events}$ **number of events**: The number of events executed by an LP during a simulation in which $I_{cp} = 1$. (This includes events rolled back.)
\(N_{\text{rollbacks}}\) number of rollbacks: The number of times an LP rolls back during a simulation in which \(I_{cp} = 1\).

\(T_{cp}\) checkpoint time: The time required to checkpoint the state of an LP. This time is assumed to be constant.

\(T_{\text{event}}\) expected event execution time: The average time required for an LP to process an event.

\(T_{\text{overhead}}\) state-saving overhead: The expected value of the time overhead associated with saving the state of an LP. This time combines the time required for checkpointing and the time required to reconstruct uncheckpointed states.

Equation 1 below gives a lower bound on the state-saving overhead associated with the checkpoint interval \(I_{cp}\). In order to arrive at this formula, the following assumptions are made: First, it is assumed that the behaviour of the system is not affected by the checkpoint interval. Second, it is assumed that the number of events recomputed during a rollback in order to reconstruct a missing state is uniformly distributed in the interval \([0, I_{cp} - 1]\).

\[
T_{\text{lower}}(I_{cp}) = \frac{N_{\text{rollbacks}}}{2} \left[ (I_{cp} - 1)T_{\text{event}} + \left( \frac{2N_{\text{events}}}{I_{cp}} \frac{N_{\text{rollbacks}}}{I_{cp}} + 1 \right) - 1 \right] T_{cp}
\]  

(1)

The checkpoint interval \(I_{cp}^+\) that minimizes \(T_{\text{lower}}\) is given by \(I_{cp}^+ = \sqrt{\alpha(2/\beta + 3)}\) where, \(\alpha = T_{cp}/T_{\text{event}}\) and \(\beta = N_{\text{events}}/N_{\text{rollbacks}} - 1\).
Equation 2 below gives an upper bound on the state-saving overhead associated with the checkpoint interval $I_{cp}$. To arrive at the upper bound, it is assumed that the number of events recomputed during a rollback in order to reconstruct a missing state is $I_{cp}$.

$$T_{upper}(I_{cp}) = N_{rollbacks} \left[ (I_{cp} - 1) T_{event} + \left( \frac{N_{events}}{N_{rollbacks}} + \frac{I_{cp} - 1}{I_{cp}} \right) T_{cp} \right] \quad (2)$$

The checkpoint interval $I_{cp}^-$ that minimizes $T_{upper}$ is given by $I_{cp}^- = \sqrt[3]{\alpha}$. The actual overhead, $T_{overhead}$, lies between $T_{upper}$ and $T_{lower}$. Therefore, the optimal checkpoint interval, $I_{cp}^{optimal}$, lies between the two roots of the equation $T_{lower}(I_{cp}) = T_{upper}(I_{cp}^-)$ provided the roots are not less than 1 (See Figure 10). There are three cases to consider, depending on where the roots fall in relation to the value 1. Table 5 gives formulae for the lower, $I_{cp}^l$, and upper, $I_{cp}^u$, bounds on the checkpoint interval for each of the three cases.

4.4.2 Empirical Results

In [13], Lin and Lazowska argue that the optimal checkpoint interval ($I_{cp}^{optimal}$) is likely to fall in the smaller interval $[I_{cp}^-, I_{cp}^+]$. In Figure 11 we plot the experimentally determined optimal checkpoint intervals vs. checkpoint time (inherent plus loop delay) for various test cases. On the same axes we have plotted curves showing $I_{cp}^l$, $I_{cp}^-$, $I_{cp}^+$, and $I_{cp}^u$ vs. checkpoint time. The latter curves were obtained by measuring the number of events ($N_{events}$), number of rollbacks ($N_{rollbacks}$), expected event execution time ($T_{event}$), and state-saving time ($T_{cp}$)
for a simulation using a checkpoint interval of one and using the equations given above.

Figure 11 shows that for the SVTF scheduling algorithm (using either lazy or aggressive cancellation) the time-optimal checkpoint interval either falls within the Lin-Lazowska interval or is slightly higher. Hence, using $I_{cp}^+$ as a predictor for the optimal checkpoint interval appears to be a good heuristic. In addition, it has already been shown that erring on the side of a checkpoint interval that is too large will degrade performance less than erring on the side of a checkpoint interval that is too small.

Figure 11 also shows that in the case of the STF scheduling algorithm, the optimal checkpoint intervals are generally larger than the Lin-Lazowska interval. Note, the theoretical analysis does not model the fact that the system behaviour (specifically, the distribution of rollback distances) will change as the checkpoint interval is changed. It appears that for the STF scheduling algorithm, this assumption is not valid (see Figure 4(c)). However, Figure 1 shows that the execution time vs. checkpoint interval curves are fairly flat near the minima. Hence, using $I_{cp}^+$ as a predictor for the optimal checkpoint interval would not incur a significant time penalty.

The difference between the theoretical and empirical results can be explained by examination of Figure 9. This figure indicates that as the checkpoint time increases, the time-optimal value of checkpoint interval also increases. It would appear that this effect would continue, were it not for the onset of thrashing. Thus the time-optimal checkpoint interval will be limited by the value of checkpoint interval that causes thrashing. It is of note that the theoretical derivation assumed that the number of rollback cycles would be constant (and thus
neither thrashing nor throttling would occur). Since the SVTF scheduling algorithm thrashes with smaller values of checkpoint interval (See Figure 8), its time-optimal checkpoint interval values are prevented from exceeding $I_{cp}$.

Since the theoretical analysis does not account for the interaction of LPs on the same processor (throttling), it also fails to account for the significant reduction in the number of rollback cycles which can result from throttling. As a result for operation in the throttling region, the time-optimal checkpoint interval may be expected to be larger than that predicted by the theoretical analysis (as seen in Figure 11). In a more detailed theoretical analysis the bounds on the time-optimal checkpoint interval are linear in the number of rollback cycles[14]. As a result, since throttling decreases the number of rollback cycles, it would be expected that the (theoretical) time-optimal checkpoint interval estimated above would underestimate the true time-optimal checkpoint interval.

5 Space vs. Checkpoint Interval

To this point, we have been primarily concerned with the effect of checkpoint interval on the execution time of a simulation. Of course, another effect of varying the checkpoint interval is to change the amount of memory used to store checkpointed state information as well as message buffers. Recall that checkpointed states and message buffers with timestamps less than GVT are fossils (garbage). In principle, such memory can be reclaimed by the fossil collection algorithm. However, all practical fossil collection algorithms lag the instan-
taneous GVT. In the current implementation of Yaddes, a circulating token GVT algorithm is used[20].

It is difficult to obtain time-averaged memory utilization data without significantly perturbing the behaviour of the simulation. However, it is relatively simple to determine the maximum memory utilization for checkpointed state information and message buffers independently, as well as for total storage (state plus message buffers). In the results reported below each processor determined separately the maximum amount of memory required for states and for messages. Following the completion of the simulation, these results were combined and the combined results are presented in the discussion below.

5.1 State Memory vs. Checkpoint Interval

Figure 12(a) shows the maximum amount of memory needed for checkpointed state information (in bytes) vs. checkpoint interval for the hypercube topology simulations using eight processors. These results are from the same simulations as shown in Figure 1. The state of an LP is 664 bytes. (Note, the curves in Figure 12 are not smooth because in each case, the datum plotted is a single sample of the extreme value of a random process sampled over a finite interval.)

Figure 12(a) clearly shows that increasing the checkpoint interval from one to four substantially decreases the maximum state storage. However, increasing the checkpoint interval beyond four has very little added benefit.
5.2 Message Memory vs. Checkpoint Interval

As the checkpoint interval is increased, less memory is required to store checkpointed states. However, an LP must be able to reconstruct any of its prior states between GVT and its current LVT. As the checkpoint interval increases, it becomes more likely that the last checkpointed state has a timestamp less than the instantaneous GVT. In other words, even though an LP will not rollback further than GVT, it may not have any checkpointed states in the interval [GVT,LVT]. Thus, it must reconstruct the state from one having a timestamp less than the instantaneous GVT. Thus, the input messages required to reconstruct the state must also be retained. (Note the fossil collection algorithm must take care not to discard such states and messages.) Consequently, as the checkpoint interval increases, the maximum storage required for message buffers also increases.

Figure 12(b) shows the maximum amount of memory needed for message buffers (in bytes) vs. checkpoint interval for the hypercube topology simulations using eight processors. These results are from the same simulations as shown in Figure 1. The size of a message buffer is 32 bytes. Figure 12(b) clearly shows that increasing the checkpoint interval generally increases the maximum amount of memory needed for message buffer storage.

5.3 Total Memory vs. Checkpoint Interval

Figure 12(c) shows maximum total storage vs. checkpoint interval for the hypercube topology simulations using eight processors. These curves show that as the checkpoint interval is
increased, maximum total storage decreases quickly to a minimum. Total storage then increases slowly with further increases in checkpoint interval. The space-optimal checkpoint interval is the checkpoint interval which minimizes the maximum total storage required for the simulation. Just as in the case of time (Section 4.2), Figure 12(c) shows that it is better to select a checkpoint interval that is too large rather than one that is too small (regardless of the checkpointing delay costs).

6 Space/Time Trade-off vs. Checkpoint Interval

Figure 13 plots maximum total storage vs. execution time. Each curve in this figure is a parametric curve obtained by varying the checkpoint interval. Each curve represents a different checkpoint time. These curves show the space/time trade-off as checkpoint interval is varied for various checkpoint times. (Note, the curves in Figure 13 are not smooth because each point plotted represents a single sample of the extreme value of a random process sampled over a finite interval.)

These curves also demonstrate the thrashing and the throttling phases. As the checkpoint interval is increased from one, (the throttling phase) maximum total storage decreases substantially while there is little effect on the total execution time. Then as the corner is turned (and the thrashing phase is entered), further increases in the checkpoint interval have little effect on the maximum total storage, but increase the execution time. The impact of thrashing is more noticeable in the SVTF scheduling algorithm (Figure 13(a)) because, as
noted in Section 4.3, thrashing occurs at smaller values of checkpoint interval than when the
STF scheduling algorithm is used.

Figure 13 also shows that as the checkpoint time is increased, the curves “close up”. I.e.,
the sensitivity of the execution time to small checkpoint intervals increases.

7 Summary

This paper has presented empirical results showing the trade-off between time and space in optimistically synchronized parallel discrete-event simulation. The simulations were im-
plemented using Yaddes on a Transputer message-passing multiprocessor. The empirical
results reported in this paper were obtained from various topologies of closed queueing net-
work simulations. This class of simulation is time invariant with respect to the load, the
number of processes, the topology of the interconnection between the processes, and the be-
haviour of the processes. Because of this, the optimum load balance can be easily achieved.
Although the specific results summarized below are relevant only to this class of problem, we
expect that changing the checkpoint interval is an important consideration for other classes
of simulation and that an understanding of the mechanisms of throttling and thrashing is
essential.

The effects of varying the checkpoint interval on the simulation execution time were
measured. Various scheduling algorithms and cancellation strategies were examined. The
empirical data was found to be in fair agreement with the analysis of Lin and Lazowska for
SVTF scheduling, but poorer for STF scheduling because the theory does not consider the
effects of multiple processes per processor.

The results presented here demonstrate two operational modes in a homogeneous time-
warp simulation problems: thrashing—where the large time penalties associated with roll-
back and coast forward cause a significant degradation in performance; and throttling—where
the increasing costs of each rollback, are beneficial to the overall simulation performance.

The execution time results also indicate that problems which are relatively unaffected
by rollbacks (in either number of rollbacks or in the severity of the effect of each rollback)
which execute in a saturated environment are promising candidates for checkpoint interval
values greater than one.

The effects of varying the checkpoint interval on the maximum total memory storage
were measured. It was shown that the maximum total storage needed could be substantially
reduced by increasing the checkpoint interval.

Finally, the trade-off between time and space was examined. It was shown that the time-
optimal and space-optimal checkpoint intervals are not necessarily the same. Furthermore,
choosing a checkpoint interval that is too small will increase space more than time; whereas,
choosing a checkpoint interval that is too large increases execution time more than space. It
would seem that using a checkpoint interval of $I_{cp}^+$ is a good heuristic.
8 Acknowledgments

We are grateful to the referees for their generous and insightful comments—their input has significantly improved this paper. We would like to thank Jim Field for his database and graph plotting package.

References


[23] PREISS, B. R., AND LOUCKS, W. M. Prediction and lookahead in distributed simulation. CCNG Technical Report E-191, Department of Electrical Engineering and
Computer Communications Networks Group, University of Waterloo, 1989.


1167.
Table Captions

Table 1: Test Cases

Table 2: Optimal Checkpoint Intervals (STF Scheduling)

Table 3: Optimal Checkpoint Intervals (SVTF Scheduling)

Table 4: Unidirectional Links Connecting Processes and Processors (processors = 8)

Table 5: Time-Optimal Checkpoint Interval Bounds
<table>
<thead>
<tr>
<th>topology</th>
<th>load</th>
<th>number of processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>hypercube</td>
<td>4</td>
</tr>
<tr>
<td>I</td>
<td>mesh</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>multistage ring</td>
<td>4</td>
</tr>
<tr>
<td>II</td>
<td>hypercube</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>hypercube</td>
<td>8</td>
</tr>
<tr>
<td>III</td>
<td>hypercube</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>hypercube</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1: Test Cases
<table>
<thead>
<tr>
<th>Checkpoint Delay Loop cancellation topology</th>
<th>Time-Optimal $I_{cp}$ 0 ms aggr. lazy</th>
<th>1 ms aggr. lazy</th>
<th>Execution 0 ms aggr. lazy</th>
<th>Time (s) 1 ms aggr. lazy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group I: topology varies</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hypercube $^b$ 4 8</td>
<td>4 2</td>
<td>8 5</td>
<td>27.1 26.9</td>
<td>29.5 29.5</td>
</tr>
<tr>
<td>mesh</td>
<td>4 3</td>
<td>6 6</td>
<td>27.9 27.6</td>
<td>29.2 29.1</td>
</tr>
<tr>
<td>multistage</td>
<td>4 2</td>
<td>5 6</td>
<td>27.7 27.5</td>
<td>29.2 29.1</td>
</tr>
<tr>
<td>ring</td>
<td>4 8</td>
<td>$\leq15$ $\leq15$</td>
<td>27.7 27.5</td>
<td>28.4 28.0</td>
</tr>
<tr>
<td>Group II: load varies</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hypercube 1 8</td>
<td>2 2</td>
<td>4 2</td>
<td>21.1 22.7</td>
<td>22.8 25.3</td>
</tr>
<tr>
<td>hypercube $^b$ 4 8</td>
<td>4 2</td>
<td>8 5</td>
<td>27.1 26.9</td>
<td>29.5 29.5</td>
</tr>
<tr>
<td>hypercube 8 8</td>
<td>3 5</td>
<td>8 8</td>
<td>29.0 28.5</td>
<td>30.8 30.8</td>
</tr>
<tr>
<td>Group III: number of processors varies</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hypercube 4 2</td>
<td>5 6</td>
<td>$\leq15$ $\leq15$</td>
<td>85.3 84.5</td>
<td>87.4 86.6</td>
</tr>
<tr>
<td>hypercube 4 4</td>
<td>4 6</td>
<td>10 $\leq15$</td>
<td>45.8 45.4</td>
<td>47.4 46.8</td>
</tr>
<tr>
<td>hypercube $^b$ 4 8</td>
<td>4 2</td>
<td>8 5</td>
<td>27.1 26.9</td>
<td>29.5 29.5</td>
</tr>
</tbody>
</table>

$^a$number of processors

$^b$italics indicate the base case (repeated)

Table 2: Optimal Checkpoint Intervals (STF Scheduling)
<table>
<thead>
<tr>
<th>Checkpoint Delay Loop cancellation</th>
<th>Time-Optimal $I_{cp}$</th>
<th>Execution</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 ms</td>
<td>1 ms</td>
<td>0 ms</td>
</tr>
<tr>
<td></td>
<td>aggr. lazy</td>
<td>aggr. lazy</td>
<td>aggr. lazy</td>
</tr>
<tr>
<td>topology</td>
<td>load</td>
<td>$P^a$</td>
<td>28.7</td>
</tr>
<tr>
<td>__________</td>
<td>______</td>
<td>______</td>
<td>______</td>
</tr>
<tr>
<td>Group I: topology varies</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>mesh</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>multistage</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>ring</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Group II: load varies</td>
<td>4</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>hypercube</td>
<td>8</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Group III: number of processors varies</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>hypercube</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>hypercube</td>
<td>4</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

*a number of processors

*bitalics indicate the base case (repeated)

Table 3: Optimal Checkpoint Intervals (SVTF Scheduling)
<table>
<thead>
<tr>
<th>Topology</th>
<th>Unidirectional Links per Node</th>
<th>Off-Processor Unidirectional Links per Processor</th>
<th>Number of Adjacent Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>ring</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>multistage</td>
<td>4</td>
<td>16 (on 2 processors) 32 (on 6 processors)</td>
<td>2 (on 2 processors) 3 (on 6 processors)</td>
</tr>
<tr>
<td>mesh</td>
<td>8</td>
<td>32</td>
<td>2</td>
</tr>
<tr>
<td>hypercube</td>
<td>12</td>
<td>48</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4: Unidirectional Links Connecting Processes and Processors (processors = 8)
<table>
<thead>
<tr>
<th>case</th>
<th>conditions</th>
<th>( I^l_{cp} )</th>
<th>( I^u_{cp} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 1 \leq I^-<em>{cp} \leq I^+</em>{cp} )</td>
<td>( a - b )</td>
<td>( a + b )</td>
</tr>
<tr>
<td>2</td>
<td>( I^-<em>{cp} \leq 1 \leq I^+</em>{cp} )</td>
<td>1</td>
<td>( \alpha(2\beta + 3) )</td>
</tr>
<tr>
<td>3</td>
<td>( I^-<em>{cp} \leq I^+</em>{cp} \leq 1 )</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

where \( a = \frac{1}{2}(3\alpha - 1) + 2\sqrt{\alpha\beta} \) and \( b = \frac{1}{2}(1 - 18\alpha - 8\sqrt{\alpha\beta} + 9\alpha^2 + 24\alpha\sqrt{\alpha\beta} + 8\alpha\beta)^{1/2} \)

Table 5: Time-Optimal Checkpoint Interval Bounds
Figure Captions

Figure 1: Execution Time vs. Checkpoint Interval, processors = 8, hypercube topology, load = 4.

Figure 2: Execution Time vs. Checkpoint Interval, processors = 8, hypercube topology, load = 4, lazy cancellation, STF scheduling.

Figure 3: Operation of Time Warp with $I_{cp} > 1$.

Figure 4: Throttling and Thrashing, processors = 8, load = 4, Lazy Cancellation, STF scheduling, loop delay = 0 ms, STF scheduling.

Figure 5: Effects of Topology on Performance, processors = 8, load = 4, lazy cancellation, STF scheduling, loop delay = 0 ms.

Figure 6: Effects of load on Performance, processors = 8, hypercube topology, lazy cancellation, STF scheduling, loop delay = 0 ms.

Figure 7: Effects of the Number of Processors on Performance, hypercube topology, load = 4, lazy cancellation, STF scheduling, loop delay = 0 ms.

Figure 8: Effects of Scheduling Policy on Performance, processors = 8, hypercube topology, load = 4, lazy cancellation, loop delay = 0 ms.

Figure 9: Effects of Checkpointing Time on Performance, processors = 8, hypercube topology, load = 4, lazy cancellation, STF scheduling.

Figure 10: State Saving Overhead vs. Checkpoint Interval.
Figure 11: Optimum Checkpoint Interval vs. Checkpoint Time, processors = 8, hypercube topology, load = 4.

Figure 12: Maximum Memory Usage vs. Checkpoint Interval, processors = 8, hypercube topology, load = 4, loop delay = 0 ms.

Figure 13: Maximum Total Storage vs. Execution Time, processors = 8, hypercube topology, load = 4, lazy cancellation.
Figure 1: Execution Time vs. Checkpoint Interval, processors = 8, hypercube topology, load = 4.
Figure 2: Execution Time vs. Checkpoint Interval, processors = 8, hypercube topology, load = 4, lazy cancellation, STF scheduling.
Figure 3: Operation of Time Warp with $I_{cp} > 1$. 

Legend:

- Checkpointed state
- Uncheckpointed state

Legend:

- Checkpointed state
- Uncheckpointed state

$I_{cp}$

straggler

First state to (potentially) alter the course of the simulation

required state

cost forward

phase

virtual time
Figure 4: Throttling and Thrashing, processors = 8, load = 4, Lazy Cancellation, STF scheduling, loop delay = 0 ms, STF scheduling.
Figure 5: Effects of Topology on Performance, processors = 8, load = 4, lazy cancellation, STF scheduling, loop delay = 0 ms.
Figure 6: Effects of load on Performance, processors = 8, hypercube topology, lazy cancellation, STF scheduling, loop delay = 0 ms.
Figure 7: Effects of the Number of Processors on Performance, hypercube topology, load = 4, lazy cancellation, STF scheduling, loop delay = 0 ms.
Figure 8: Effects of Scheduling Policy on Performance, processors = 8, hypercube topology, load = 4, lazy cancellation, loop delay = 0 ms.
Figure 9: Effects of Checkpointing Time on Performance, processors = 8, hypercube topology, load = 4, lazy cancellation, STF scheduling.
Figure 10: State Saving Overhead vs. Checkpoint Interval.
Figure 11: Optimum Checkpoint Interval vs. Checkpoint Time, processors = 8, hypercube topology, load = 4.
Figure 12: Maximum Memory Usage vs. Checkpoint Interval, processors = 8, hypercube topology, load = 4, loop delay = 0 ms.
Figure 13: Maximum Total Storage vs. Execution Time, processors = 8, hypercube topology, load = 4, lazy cancellation.