On the Trade-off between Time and Space in Optimistic Parallel Discrete-Event Simulation

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

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

Abstract

Optimistically synchronized parallel discrete-event simulation is based on the use of communicating sequential processes. Optimistic synchronization means that the processes execute under the assumption that synchronization is fortuitous. Periodic checkpointing of the state of a process allows the process to roll back to an earlier state when synchronization errors occur. This paper examines the effects of varying the frequency of checkpointing on the time and space needed to execute a simulation.

The results presented in this paper were obtained from the simulation of closed stochastic queueing networks with several different topologies. Various process scheduling algorithms and message cancellation strategies are considered. The empirical results are compared with analytical formulae predicting time-optimal checkpoint intervals. 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.

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 periodic checkpointing of states and by reconstruction of the desired state from a checkpointed one. Hence, there is a time-penalty associated with the reconstruction of a state and a space-penalty associated with the checkpointing of a state.

The next section introduces parallel discrete event simulation and describes the implementation of the most common optimistic synchronization method. It also covers process scheduling, message cancellation, and introduces the checkpoint interval. In Section 3, the analytical results of Lin and Lazowska which predict time-optimal checkpoint intervals are presented. Section 4 describes the experiments performed to assess the space/time trade-off. The experimental results are presented in Sections 5, 6, and 7. Section 5 examines the effects of checkpoint interval on total execution time. Section 6 shows the effects of checkpoint interval on the maximum memory requirements of a simulation. Section 7 shows the trade-off between space and time. Finally, we summarize our observations in Section 8.

2. Parallel Discrete-Event Simulation

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.

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[9]. (In all cases in this paper, *timestamp* refers to the receive time of a message.) 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*[17]. 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 and the techniques for avoiding or detecting and recovering from deadlock are costly[2]. 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 a state immediately prior to the violation (*rollback*), and execution resumes[5].

In this paper we focus on the most common optimistic synchronization method, namely Time-Warp[4]. Time-Warp facilitates optimistic synchronization in the following manner: Conceptually, each LP has an input queue of unprocessed messages (*future queue*) in which all of the messages have timestamps greater than or equal to the LVT, and a queue of processed input messages (*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 that 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 pending antimessage and the new output message will annihilate.

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 (often referred to as *checkpointing*). Checkpointing has two adverse effects on the simulation: memory is consumed, and it is possible that there will be insufficient memory to permit the simulation to complete in any amount of time; in addition, processing time is needed to copy the portion of the state needing to be checkpointed. There are several ways that these ''overheads'' (time and space) can be addressed in an optimistic simulation.

2.3.1. Checkpoint Interval

One approach to reducing the state-saving overhead is to reduce its frequency. 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 $n>1$. 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.

If a rollback occurs when $I_{cp}=1$, then the required state of the LP can be restored directly from the state queue. This is because the state queue contains every possible state to which a rollback may occur. 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 rollback to an earlier checkpointed state and recompute the desired state from the earlier state. The desired state is recomputed by reprocessing input messages. Note that one of the penalties associated with $I_{cp}>1$ is the time required to reconstruct the uncheckpointed states. Other costs are discussed in Section 6.3.

An important consideration when implementing Time-Warp with $I_{cp}>1$ is how to reconstruct the uncheckpointed state if needed. Although the state is easily reconstructed by reprocessing input messages, output messages regenerated during state reconstruction cannot be sent. The reconstruction 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.

2.3.2. 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.

2.3.3. Global Virtual Time Overhead

Global virtual time (GVT) is an important concept in optimistic parallel simulation. GVT is the minimum of the LVTs of all LPs and the timestamps of any messages in transit. Since no LP will ever rollback to a time before the GVT, checkpointed states and messages with timestamps less than GVT 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 [13].

2.4. 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. In this paper we examine three different scheduling algorithms. For a more complete discussion of scheduling algorithms see [1] and [6]. All three algorithms are non-preemptive.* I.e., once an LP begins to process an input message, the processing of that message will proceed to completion before another LP will execute. Furthermore, the arrival of an antimessage at an LP will not terminate the current processing cycle of that LP.

2.4.1. Round-Robin Scheduling

When using the round-robin (RR) scheduling algorithm, LPs that are ready to execute are allowed to process input messages in round-robin fashion one at a time.** An LP is ready to execute as long as it has input messages to process. 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.

This scheduling algorithm is easily implemented using a list of process descriptors. Although this algorithm is easy to implement, it fails to take into account the temporal interaction of events and LPs.

---

* Note that non-preemptive Time Warp scheduling algorithms are, by their nature, incorrect, as they can allow a correct simulation to get trapped in an infinite loop during an optimistic execution. The loop can only be terminated if the event can be interrupted by the rollback that undoes this event.

** 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 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.

2.4.2. Minimum-Virtual-Time Scheduling

In order to ensure steady progress of the simulation, it seems intuitive to give higher priority to LPs with lower LVTs. The minimum-virtual-time (MVT) scheduling algorithm always chooses the (ready) LP with the smallest LVT. The chosen LP is allowed to process one input message. In order to efficiently select the LP with the smallest LVT, the process descriptors of the LPs can be maintained in a binary heap data structure. (Furthermore, if the number of processes per processor has a fixed upper bound, the binary heap can be stored in a linear array.)

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

2.4.3. Minimum-Message-Timestamp Scheduling

The minimum-message-timestamp (MMT) scheduling algorithm gives higher priority to LPs having input messages with lower timestamps. This algorithm always chooses the LP with input message having the smallest timestamp. Scheduling by smallest message timestamp tends to reduce the number of rollbacks for the the experiments discussed in Section 4. 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). As in the case of the MVT algorithm, this algorithm can be implemented by storing the process descriptors in a binary heap data structure implemented as a linear array.

3. Time-Optimal Checkpoint Intervals

In [7], Lin and Lazowska derive 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 [7] 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 checkpoints.

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.)

$N_{rollbacks}$ — *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_{event} — *expected event execution time*: The average time required for an LP to process an event.

$T_{overhead}$ — *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.

Lower bound on state-saving overhead: The inequality below gives a lower bound on the state-saving overhead associated with the checkpoint interval I_{cp}:

$$T_{overhead} \geq T_{lower}(I_{cp})$$

where

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

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, to arrive at the lower bound, it is assumed that the number of events recomputed during a rollback is uniformly distributed in the interval $[0, I_{cp}-1]$. Note, this bound can be computed from parameters measured from a simulation performed with I_{cp}=1.

The checkpoint interval that maximizes the right-hand side of the inequality above is given by

$$I_{cp}^{+} = \left\lceil \sqrt{\alpha(2\beta+3)} \right\rceil \quad (2)$$

where,

$$\alpha = T_{cp}/T_{event}\ ,\ \beta = N_{events}/N_{rollbacks} - 1$$

Upper bound on state-saving overhead: The inequality below gives an upper bound on the state-saving overhead associated with the checkpoint interval I_{cp}:

$$T_{overhead} \leq T_{upper}(I_{cp})$$

where

$$T_{upper}(I_{cp}) = N_{rollbacks} \times \left[(I_{cp}-1)T_{event} + \left(\frac{N_{events}/N_{rollbacks}+(I_{cp}-1)}{I_{cp}} \right) T_{cp} \right] \quad (3)$$

To arrive at the upper bound, it is assumed that the number of events recomputed during a rollback is I_{cp}. The checkpoint interval that minimizes the right-hand side of the inequality above is given by

$$I_{cp}^{-} = \left\lceil \sqrt{\alpha\beta} \right\rceil . \quad (4)$$

It can be shown that the optimal checkpoint interval, $I_{cp}^{optimal}$, lies between the two roots of the following equation:

$$T_{lower}(I_{cp}) = T_{upper}(I_{cp}^{-}) . \quad (5)$$

Let I_{cp}^{l} and I_{cp}^{u} represent the roots of equation (5) such that $I_{cp}^{l} \leq I_{cp}^{u}$. The roots of equation (5) are given by the following equations:

$$I_{cp}^{l} = a - b,\ I_{cp}^{u} = a + b \quad (6)$$

$$a = \tfrac{1}{2}(3\alpha - 1) + 2\sqrt{\alpha\beta}$$

$$b = \tfrac{1}{2}(1 - 18\alpha - 8\sqrt{\alpha\beta} + 9\alpha^2 + 24\alpha\sqrt{\alpha\beta} + 8\alpha\beta)^{1/2}$$

Note that $I_{cp}^{l} \leq I_{cp}^{-} \leq I_{cp}^{+} \leq I_{cp}^{u}$. In [7], Lin and Lazowska suggest the use of the smaller interval, $[I_{cp}^{-}, I_{cp}^{+}]$, to estimate the optimal checkpoint interval because its bounds are easier to calculate than I_{cp}^{l} and I_{cp}^{u} and because their simulations showed that it is likely for $I_{cp}^{optimal}$ to fall in the smaller interval.

4. Experimental Overview

The results presented in this paper were obtained from parallel simulations of closed stochastic queueing network benchmarks based on those described in [10]. The following is a very brief description of the benchmarks. For more detailed descriptions see [8, 12, 14, 15]. Each simulation was implemented in three ways (discussed in Section 4.2). These implementations were coded and run on a transputer system (Section 4.3), and a number of experiments were performed on the system (Section 4.4).

4.1. 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 μ_{min}; the mean service time is $\mu=5\mu_{min}$; and, the system was simulated for 1000μ time steps.

The suite of benchmarks includes four network topologies—ring ($f_{in}=f_{out}=2$), multistage ($f_{in}=f_{out}=2$), 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.

4.2. Logical Process Characteristics

The stochastic queueing network described above could be implemented in several different ways. In order to increase the generality of the results presented here we have used three different implementations. (Although, for this exact application one implementation is more efficient.)

Each node in the network is modelled by a single LP. Three different implementations of the LP have been studied. Each LP implementation provides a different degree of *lookahead*[3]. The lookahead of an LP is a measure of predictiveness and is defined as follows: Consider an LP whose LVT is t_{commit} when it sends a message with timestamp t_{effect}. I.e., the LP makes a prediction when its LVT is t_{commit} that a customer will depart at time t_{effect}. The lookahead of an LP is simply $E[t_{effect}-t_{commit}]$, where the expectation is taken over all output messages. *Lookahead ratio* is defined as the expected time a customer will spend at a node divided by the lookahead. I.e., lookahead ratio is $E[t_{effect}-t_{cause}]/E[t_{effect}-t_{commit}]$, where t_{cause} is the time at which that customer arrives.

Table I summarizes the event execution times and the state sizes for the systems simulated.

Table I
LP Characteristics

lookahead ratio	load	state size (bytes)	T_{event} (ms)
1	1	664	2.2
	4		2.9
	8		3.5
4	4	448	1.7
40	4	412	1.2

In the results discussed below, lookahead ratio 1, load 4 is used as the base case against which the others are compared. Of note in this table is the state size. The state includes: a queue of unprocessed messages (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 other implementations (lookahead 4 and 40) were coded differently and represent other valid solutions to the user’s problem.

4.3. Simulation Software and Hardware

The simulations were implemented using the Yaddes distributed discrete event simulation language[11, 13, 16]. 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.

4.4. Experiments

A total of 3240 simulations were run. Each simulation has only been run once. Therefore, it is not possible to compute confidence intervals. In addition, some of the data is ‘‘noisy.’’ This noise manifests itself as bumps in the curves.

As the base simulation, we selected the hypercube topology, an average of four customers per node (i.e., load=4), lookahead ratio of one, using eight processors. We then independently varied the parameters topology, load, lookahead ratio, and number of processors used. Table II 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; in group III, lookahead is varied; and in group IV, the number of processors used to perform the simulation is varied.

Table II
Test Cases

	topology	load	lookahead ratio	number of processors
base	hypercube	4	1	8
I	mesh	4	1	8
	ring	4	1	8
	multistage	4	1	8
II	hypercube	1	1	8
	hypercube	8	1	8
III	hypercube	4	40	8
	hypercube	4	4	8
IV	hypercube	4	1	2
	hypercube	4	1	4

Each of the test cases listed in Table II 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—RR, MVT, MMT. 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.

5. 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 and the maximum storage required to perform the simulation.

5.1. 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 MMT scheduling algorithm. When using the MVT scheduling algorithm the simulation takes between 5 and 30 percent longer than when using the MMT 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 MVT and MMT algorithms. (The time-optimal checkpoint intervals are tabulated in a later section.)

As Figure 1 shows, the execution time vs. checkpoint interval curves for the MVT and MMT 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 6).

Finally, Figure 1 shows the effect of using lazy vs. aggressive cancellation is negligible for the MVT and MMT 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.

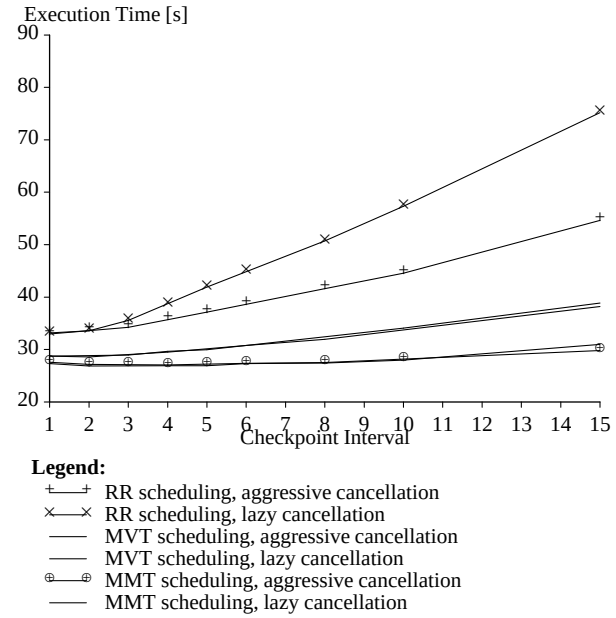

Figure 1. Execution Time vs. Checkpoint Interval, processors=8, hypercube topology, load=4, lookahead ratio=1, loop delay=0 s.

5.2. Effects of Checkpoint Time

Figure 2 shows execution time vs. checkpoint interval for the hypercube topology simulations using eight processors. In all cases, the MMT 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 frequency decreases.

Figure 2 also shows that the execution time increases more quickly for checkpoint intervals that are smaller than the optimum than for checkpoint intervals greater than the optimum. Thus, 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 frequently.

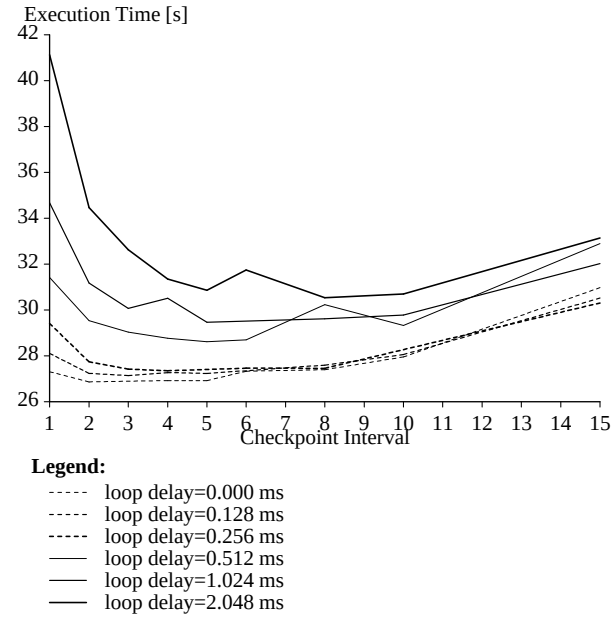

Figure 2. Execution Time vs. Checkpoint Interval, processors=8, hypercube topology, load=4, lookahead ratio=1, MMT scheduling, lazy cancellation.

5.3. Time-Optimal Checkpoint Intervals

Table III shows the experimentally determined time-optimal checkpoint intervals for the ten simulation test cases listed in Table II for each scheduling algorithm and cancellation strategy. This table was 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. Table III gives the results for the inherent checkpoint time (i.e.,

loop delay zero) simulations. Table IV gives the results for the same set of simulation cases but with a loop delay of 1 ms.

Table III
Time-Optimal Checkpoint Intervals
Zero Loop Delay

		scheduling		RR		MVT		MMT	
		cancellation		aggr.	lazy	aggr.	lazy	aggr.	lazy
topology	load	*LAR*	*P*						
hypercube	4	1	8	1	1	2	2	4	2
mesh	4	1	8	1	1	2	1	4	3
ring	4	1	8	2	2	3	2	10	8
multistage	4	1	8	2	1	2	2	3	2
hypercube	1	1	8	1	2	1	3	2	2
hypercube	8	1	8	1	1	2	2	3	5
hypercube	4	40	8	1	1	1	2	2	2
hypercube	4	4	8	1	1	2	2	3	2
hypercube	4	1	2	2	2	1	1	5	6
hypercube	4	1	4	1	1	1	1	4	6

LAR=lookahead ratio
P=number of processors

Table IV
Time-Optimal Checkpoint Intervals
1 ms Loop Delay

		scheduling		RR		MVT		MMT	
		cancellation		aggr.	lazy	aggr.	lazy	aggr.	lazy
topology	load	*LAR*	*P*						
hypercube	4	1	8	3	3	3	4	8	5
mesh	4	1	8	2	2	3	3	6	6
ring	4	1	8	4	3	5	6	†	†
multistage	4	1	8	3	2	4	4	5	6
hypercube	1	1	8	2	2	2	4	4	2
hypercube	8	1	8	3	2	4	5	8	8
hypercube	4	40	8	3	3	3	4	4	6
hypercube	4	4	8	2	3	4	5	6	6
hypercube	4	1	2	2	2	3	4	†	†
hypercube	4	1	4	2	2	3	4	10	†

† $I_{cp}^{optimal} \geq 15$
LAR=lookahead ratio
P=number of processors

Effects of Topology: An important characteristic of the various topologies is their fanout. The fanout of the ring and multistage topologies is two; the fanout of the mesh topology is four; and the fanout of the hypercube topology is six. Table III shows that increasing the fanout of the topology causes the optimal checkpoint interval to decrease. This observation is true in all cases, but more pronounced when aggressive cancellation is used. The latter observation suggests the following explanation for the fanout effect: When rollbacks occur, LPs having larger fanouts are likely to cause secondary rollbacks in more LPs than LPs having small fanouts. As the frequency of rollbacks increases, it makes more sense to checkpoint more often in order to eliminate the time needed to reconstruct uncheckpointed states. On the other hand, if the fanout is low and the rollback frequency is low, as in the case of the ring topology using MMT scheduling, the time-optimal checkpoint interval can be relatively large.

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. Results for simulations with loads of 1, 4, and 8 are given in Table III. In most cases, the table shows the expected result when using lazy cancellation that for smaller load, the optimal checkpoint interval is 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.

Effects of Lookahead: The lookahead of an LP is a measure of its predictiveness. Results for simulations with lookahead ratios of 1, 4, and 40 are given in Table III. Recall that more predictive LPs have smaller lookahead ratios and that a lookahead ratio of one corresponds to the case where the departure of a customer from an LP is predicted at the time that the customer arrives at that LP. The data in Table III indicates that the more predictive LPs (i.e., smaller lookahead ratios) need to checkpoint less frequently (i.e., larger checkpoint intervals). This is because more predicative LPs are less likely to roll back. To see why this is the case, consider an LP with lookahead ratio of one. When a customer arrives at such an LP, its departure is predicted and a message is sent without advancing the LVT of the LP. On the other hand, a less predictive LP must advance its LVT before the output message can be sent. If a rollback is required to an intermediate time value, the latter LP must perform cancellation whereas the former LP does not.

Effects of Number of Processors: The data for the MMT scheduling algorithm show 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 one processor, the MMT algorithm behaves exactly like a sequential simulation using an event list. Since a sequential simulation never rolls back, the optimum checkpoint interval is infinite. As the number of processors is increased, MMT behaves less like a sequential simulation, and the optimum checkpoint interval decreases.

5.4. Comparison of Theoretical Bounds and Experimental Results

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. As discussed above, equations (2) and (4) define the interval $[I_{cp}^{+},I_{cp}^{-}]$ in which the optimal checkoint interval ($I_{cp}^{optimal}$) is likely to fall. In Figure 3 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 substituting the measured values into equations (2), (4), and (6).

Figure 3 shows that for both the RR scheduling and MVT scheduling algorithms and for both lazy and 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 3 also shows that in the case of the MMT 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 MMT algorithm, this assumption is not valid. However, Figure 1 shows that the execution time vs. checkpoint interval curves are fairly flat. Hence, using I_{cp}^{+} as a predictor for the optimal checkpoint interval would not incur a significant time penalty.

6. 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 instantaneous GVT. In the current implementation of Yaddes, a circulating token GVT algorithm is used[13].

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).

6.1. State Memory vs. Checkpoint Interval

Figure 4 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 4 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 4 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.

6.2. 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 will also increase.

Figure 5 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 and Figure 4. The size of a message buffer is 32 bytes. Figure 5 clearly shows that increasing the checkpoint interval generally increases maximum amount of memory needed for message buffer storage.

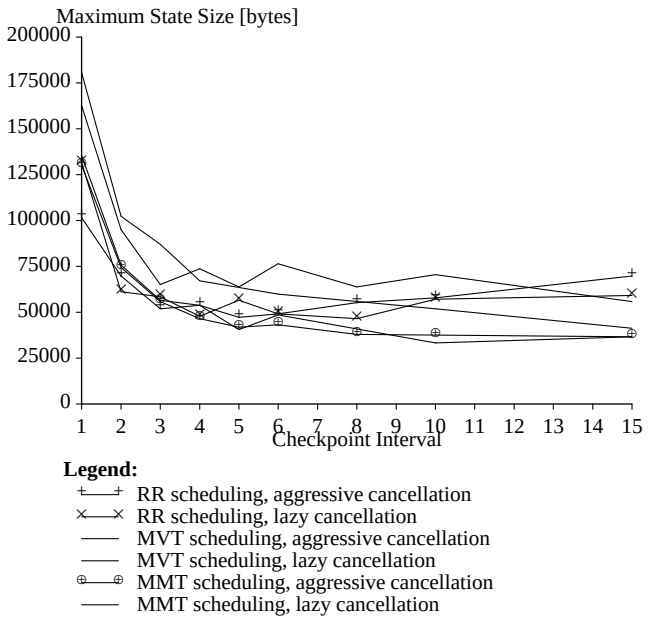

Figure 4. Maximum State Size vs. Checkpoint Interval, processors=8, hypercube topology, load=4, lookahead ratio=1, loop delay=0 s.

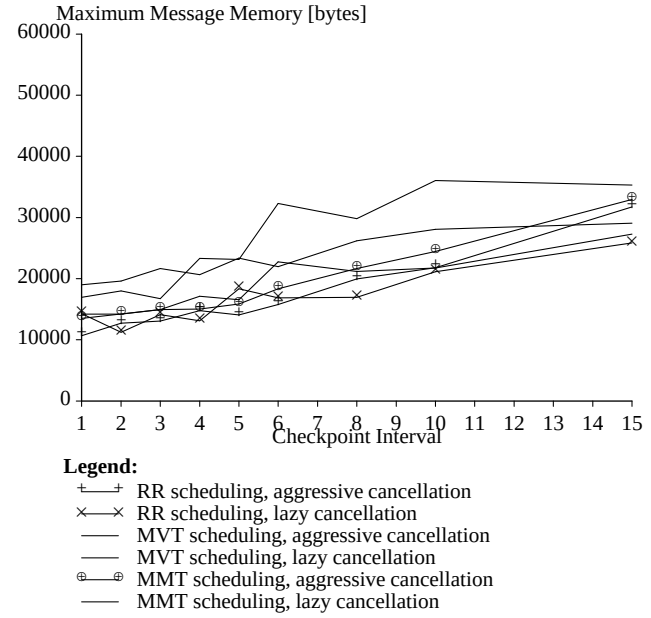

Figure 5. Maximum Message Memory vs. Checkpoint Interval, processors=8, hypercube topology, load=4, lookahead ratio=1, loop delay=0 s.

6.3. Total Memory vs. Checkpoint Interval

Figure 6 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. Figure 6 shows that it is better to select a checkpoint interval that is too large rather than one that is too small.

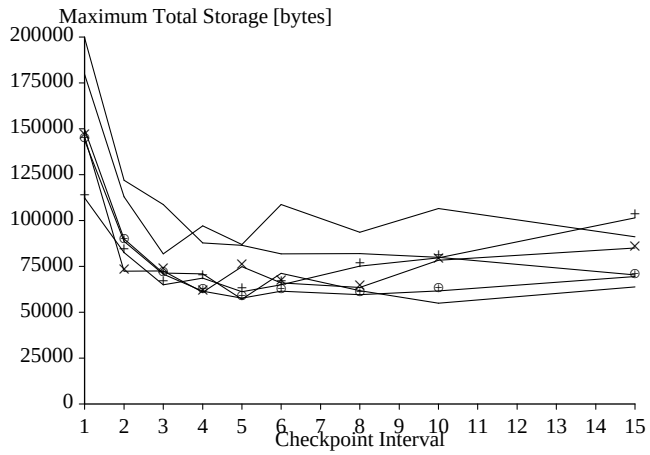

Figure 6. Maximum Total Storage vs. Checkpoint Interval, processors=8, hypercube topology, load=4, lookahead ratio=1, loop delay=0 s.

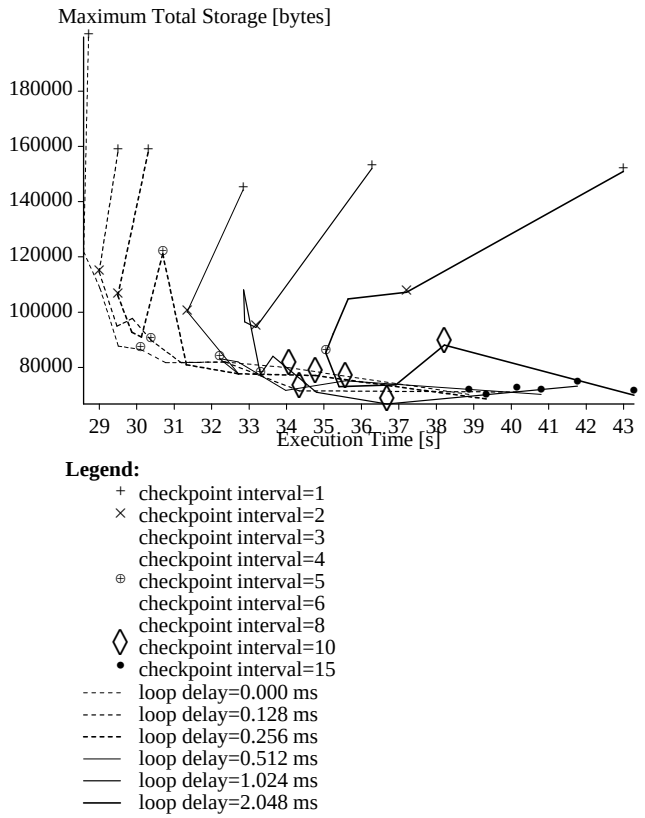

Figure 7. Maximum Total Storage vs. Execution Time, processors=8, hypercube topology, load=4, lookahead ratio=1, lazy cancellation, MVT scheduling.

7. Space/Time Trade-off vs. Checkpoint Time

Figure 7 plots maximum total storage vs. execution time. Each curve in Figure 7 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.

For the smaller checkpoint times the curves have two parts. As the checkpoint interval is increased from one, maximum total storage decreases substantially while there is little effect on the total execution time. Then, as the corner is turned, further increases in the checkpoint interval have little effect on the maximum total storage, but substantially increase the execution time.

Figure 7 also shows that as the checkpoint time is increased, the curves ''close up''. I.e., the sensitivity of the execution time to the checkpoint interval increases.

Figures 7 and 3 can be related in the following fashion: For example, consider the case where the loop delay is 0.512 *ms*. Reading the middle graph in the bottom row of Figure 3 shows that *Icpplus*=3. This corresponds to the point on Figure 7 on the thin, solid line indicated by the inverted triangle. We see that this is the left-most extremum of the thin, solid line. In this case, I_{cp}^{+} has correctly predicted the time-optimal checkpoint iterval.

8. Summary

This paper has presented empirical results showing the trade-off between time and space in optimistically synchronized parallel discrete-event simulation. The benchmarks used to generate these results consisted of simulations of closed stochastic queueing networks with various topologies. The simulations were implemented using Yaddes on a Transputer multiprocessor.

First, 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 good agreement with the analysis of Lin and Lazowska.

Second, 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 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 increases 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.

9. References

1. Burdorf, C. and Marti, J., ''Non-Preemptive Time Warp Scheduling Algorithms,'' *Operating Systems Review*, Vol. 24, No. 2, April 1990. pp. 7-18,

2. Chandy, K. M. and Misra, J., ''Distributed Simulation: A Case Study in Design and Verification of Distributed Programs,'' *IEEE Trans. on Software Engineering*, Vol. SE-5, No. 5, September 1979. pp. 440-452,

3. Fujimoto, R. M., ''Lookahead in Parallel Discrete Event Simulation,'' *Proc. 1988 Int. Conf. on Parallel Processing*, Vol. III, August 1988. pp. 34-41,

4. Jefferson, D., Beckman, B., Wieland, F., Blume, L., DiLoreto, M., Hontalas, P., Laroche, P., Sturdevant, K., Tupman, J., Warren, V., Wedel, J., Younger, H., and Bellenot, S., ‘‘Distributed Simulation and the Time Warp Operating System,’’ *Proc. 12th SIGOPS — Symposium on Operating Systems Principles*, 1987. pp. 77-93,

5. Jefferson, D. R., ‘‘Virtual Time,’’ *ACM Trans. on Programming Languages and Systems*, Vol. 7, No. 3, July 1985. pp. 404-425,

6. Lin, Y.-B. and Lazowska, E. D., ‘‘Processor Scheduling for Time Warp Parallel Simulation,’’ Technical Report 90-03-03, Dept. of Comp. Sci. and Eng., University of Washington, Seattle, Washington, March 1990.

7. Lin, Y.-B. and Lazowska, E. D., ‘‘Reducing the State Saving Overhead for Time Warp Parallel Simulation,’’ Technical Report 90-02-03, Dept. of Comp. Sci. and Eng., University of Washington, Seattle, Washington, February 1990.

8. Loucks, W. M. and Preiss, B. R., ‘‘The Role of Knowledge in Distributed Simulation,’’ *Proc. SCS Eastern Multiconf. — Distributed Simulation*, Society for Computer Simulation, January 1990. pp. 9-16,

9. Misra, J., ‘‘Distributed Discrete-Event Simulation,’’ *ACM Computing Surveys*, Vol. 18, No. 1, March 1986. pp. 39-66,

10. Nicol, D. M., ‘‘High Performance Parallelized Discrete Event Simulation of Stochastic Queueing Networks,’’ *Proc. 1988 Winter Simulation Conf.*, Society for Computer Simulation, December 1988. pp. 306-314,

11. Preiss, B. R., Loucks, W. M., and Hamacher, V. C., ‘‘A Unified Modeling Methodology for Performance Evaluation of Distributed Discrete Event Simulation Mechanisms,’’ *Proc. 1988 Winter Simulation Conf.*, Society for Computer Simulation, December 1988. pp. 315-324,

12. Preiss, B. R. and Loucks, W. M., ‘‘Prediction and Lookahead in Distributed Simulation,’’ CCNG Technical Report E-191, Computer Communications Networks Group, University of Waterloo, Waterloo, Ontario, Canada, 1989.

13. Preiss, B. R., ‘‘The Yaddes Distributed Discrete Event Simulation Specification Language and Execution Environments,’’ *Proc. SCS Eastern Multiconf. — Distributed Simulation*, Vol. 21, No. 2, Society for Computer Simulation, March 1989. pp. 139-144,

14. Preiss, B. R. and Loucks, W. M., ‘‘The Impact of Lookahead on the Performance of Conservative Distributed Simulation,’’ *Proc. SCS European Multiconf. — Modelling and Simulation*, Society for Computer Simulation, June 1990. pp. 204-209,

15. Preiss, B. R., ‘‘Performance of Discrete Event Simulation on a Multiprocessor Using Optimistic and Conservative Synchronization,’’ *Proc. 1990 Int. Conf. on Parallel Processing*, August 1990. pp. 218-222,

16. Preiss, B. R. and MacIntyre, I. D., ‘‘YADDES — Yet Another Distributed Discrete Event Simulator: User Manual,’’ CCNG Technical Report E-197, Computer Communications Networks Group, University of Waterloo, Waterloo, Ontario, Canada, 1990.

17. Reynolds, P. F., ‘‘A Spectrum of Options for Parallel Simulation,’’ *Proc. 1988 Winter Simulation Conf.*, Society for Computer Simulation, December 1988. pp. 325-332,

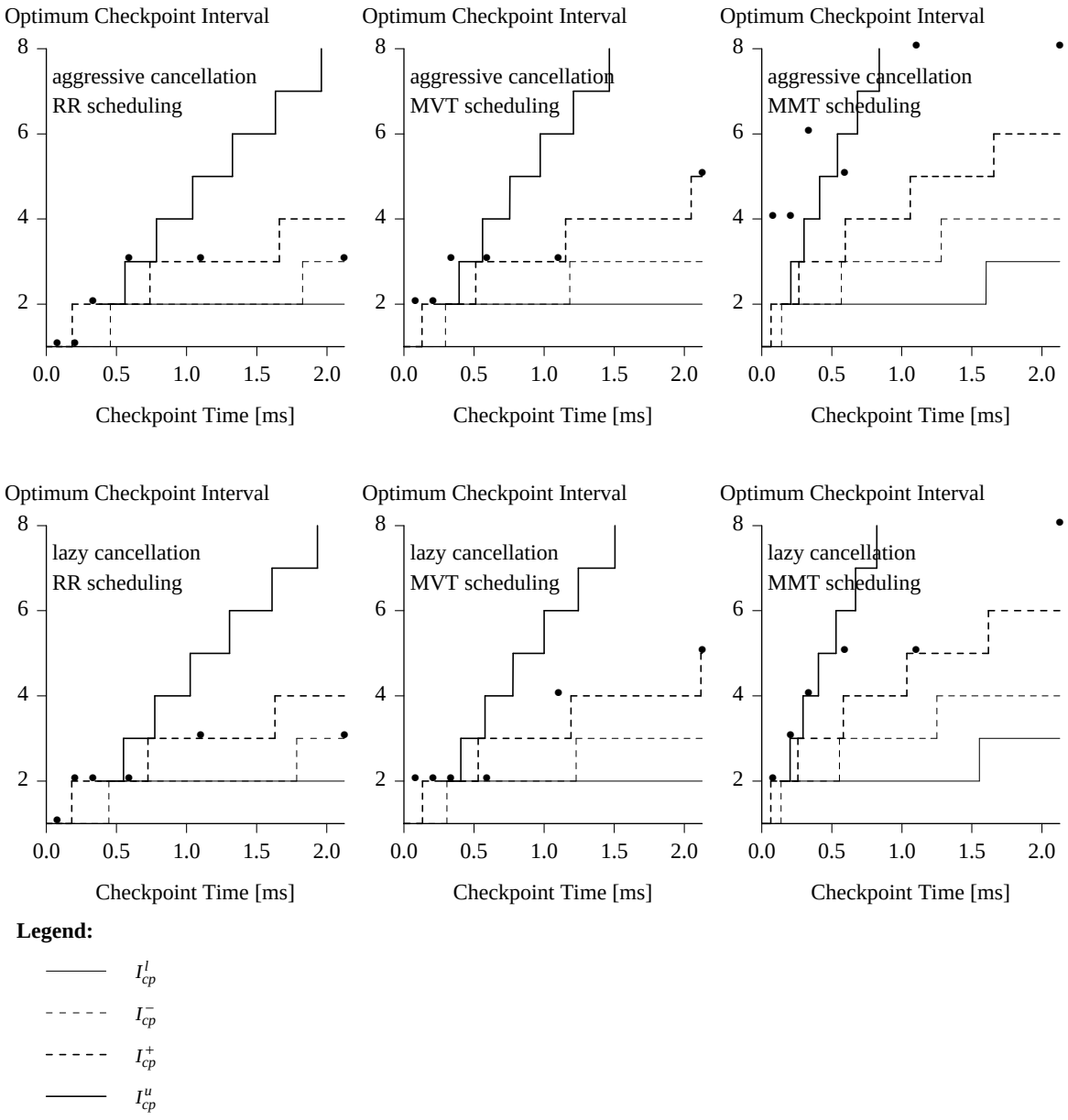

Figure 3. Optimum Checkpoint Interval vs. Checkpoint Time, processors=8, hypercube topology, load=4, lookahead ratio=1.