TO APPEAR: 12TH SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS, PRINCETON, NJ
OCTOBER 6-8, 1993

Lazy Checkpoint Coordination for Bounding Rollback Propagation

Yi-Min Wang and W. Kent Fuchs
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

Abstract

In this paper, we propose the technique of lazy checkpoint coordination which preserves process autonomy while employing communication-induced checkpoint coordination for bounding rollback propagation. The notion of laziness is introduced to control the coordination frequency and allow a flexible trade-off between the cost of checkpoint coordination and the average rollback distance. Worst-case overhead analysis provides a means for estimating the extra checkpoint overhead. Communication trace-driven simulation for several parallel programs is used to evaluate the benefits of the proposed scheme.

1 Introduction

Uncoordinated checkpointing [1–3] for parallel and distributed systems allows maximum process autonomy and independent design of recovery capability for each process. However, in a general nondeterministic execution, cascading rollback propagation may result in the *domino effect* [4] which can prevent progression of the recovery line. It has been shown that message reordering [5] and message logging [3] can effectively *reduce* rollback propagation. In order to entirely *eliminate* the possibility of domino effects, extra checkpoints need to be taken based on the communication history. Kim *et al.* [6] and Venkatesh *et al.* [7] employ transitive dependency tracking and insert a checkpoint before processing any message that introduces a new dependency. Russel [8] proves that, by inserting a checkpoint between every pair of consecutive `send` and `receive` events (in that order), domino-free recovery is ensured. The log-based approach [9–17] assumes the piecewise deterministic execution model [12] where a process execution consists of a number of deterministic state intervals, each started by a nondeterministic event. It has been shown that logging a nondeterministic event equivalently places a *logical checkpoint* [18] at the *end* of the ensuing state interval, and these extra logical checkpoints serve to eliminate the domino effect.

Coordinated checkpointing achieves domino-free recovery by sacrificing a certain degree of process autonomy and incurring run-time and extra message overhead. Usually, whenever a checkpoint is initiated by one process, all the other processes are informed and required to take appropriate checkpoints in order to guarantee the resulting set of checkpoints is consistent [19–24].

We will use the term *eager checkpoint coordination* for the coordination action performed when checkpoints are *initiated*, as described above. In contrast, processes in a system with *lazy checkpoint coordination* only coordinate their corresponding checkpoints when message communication indicates a violation of checkpoint consistency. Briatico *et al.* [25] force the receiver of a message m to take a checkpoint before processing m if the sender's checkpoint interval number tagged on m is greater than that of the receiver. Checkpoints with the same ordinal numbers are therefore always guaranteed to be consistent. However, the run-time overhead may be high due to the possibly excessive number of extra induced checkpoints. In this paper, we generalize the concept of communication-induced checkpoint coordination by introducing the notion of *laziness* Z as a measure of the frequency for performing coordination. Only corresponding checkpoints with ordinal numbers nZ, where n is an integer, are required to be consistent with each other for bounding rollback propagation. Overhead analysis shows that our generalization can significantly reduce the number of extra checkpoints compared to the previous work [25] which corresponds to the case of $Z = 1$.

2 Checkpointing and Rollback Recovery

The system considered in this paper consists of a number of concurrent processes for which all process communication is through message passing. Processes are assumed to run on fail-stop processors [26] and, for the purpose of presentation, each process is considered an individual *recovery*

[1] This research was supported in part by the Department of the Navy and managed by the Office of the Chief of Naval Research under Contract N00014-91-J-1283, and in part by the National Aeronautics and Space Administration (NASA) under Grant NASA NAG 1-613, in cooperation with the Illinois Computer Laboratory for Aerospace Systems and Software (ICLASS).

unit. In order to allow general nondeterministic execution, we do not assume a piecewise deterministic model. This implies whenever the sender of a message m rolls back and *unsends* m, the receiver which has already processed m must also roll back to undo the effect of m because the potential nondeterminism preceding the sending of m may prevent the same message from being resent during reexecution. Let $c_{i,x}$ denote the xth checkpoint ($x \geq 0$) of process p_i ($0 \leq i \leq N-1$), where N is the number of processes in the system. Two checkpoints $c_{i,x}$ and $c_{j,y}$ are then considered *inconsistent* if there is any message sent after $c_{j,y}$ and processed before $c_{i,x}$, or vice versa. In contrast, when the receiver of a message m' rolls back and *unreceives* m', the sender needs not roll back to *unsend* m' if m' can be retrieved from a message log [3, 11, 12, 27] or through a reliable end-to-end transmission protocol [14, 22].

During normal execution, each process periodically and independently saves its state as a *checkpoint* on stable storage. The interval between $c_{i,x}$ and $c_{i,x+1}$ is called the xth *checkpoint interval* of p_i. Each message is tagged with the current checkpoint interval number and the process number of the sender, and each receiver p_i performs *direct dependency tracking* [1, 28] as follows: if a message sent from (j, y) is processed in (i, x), the direct dependency of $c_{i,x+1}$ on $c_{j,y}$ is recorded.

A garbage collection procedure can be periodically invoked by any process p_i. First, p_i collects the direct dependency information from all the other processes to construct the *checkpoint graph* [1] as shown in Fig. 1(b). Then the *rollback propagation algorithm* (Fig. 2) is applied to the checkpoint graph to determine the *global recovery line*[2] (black vertices), before which all the checkpoints are obsolete and can be discarded. Alternatively, an *optimal garbage collection algorithm* [29] can be used to minimize the space overhead by discarding all the garbage checkpoints marked "X" in Fig. 1(b).

When any process initiates a rollback, it starts a similar procedure for recovery. The current volatile states of the surviving processes are treated as additional *virtual checkpoints* [2] for constructing an *extended checkpoint graph* of which the recovery line is called the *local recovery line* (shaded vertices) and indicates the consistent rollback state.

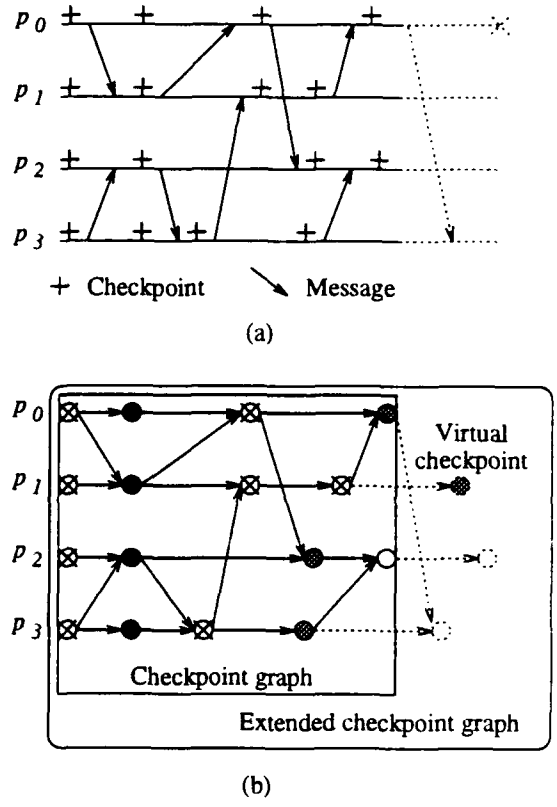


Figure 1: Checkpointing and rollback recovery. (a) example checkpoint and communication pattern; (b) checkpoint graph and extended checkpoint graph when p_0 initiates a rollback.

```
/* CP represents a checkpoint */
/* Initially, all the CPs are unmarked */
Include the latest CP of each process in the root
    set;
Mark all CPs strictly reachable from any CP in
    the root set;
While (at least one CP in the root set is marked)
    {
  Replace each marked CP in the root set by the
    latest unmarked CP of the same process;
  Mark all CPs strictly reachable from any CP
    in the root set }
The root set is the recovery line.
```

Figure 2: The rollback propagation algorithm.

3 Lazy Checkpoint Coordination

3.1 Motivation

We will refer to the checkpoints initiated independently by each process as *basic checkpoints* and those triggered by

[2]The global recovery line is to be used when the entire system fails, while a local recovery line is computed when only a subset of processes becomes faulty.

the communication as *induced checkpoints*. Fig. 3(a) illustrates a situation where the communication pattern renders most of the basic checkpoints useless for rollback recovery and the global recovery line stays at the very beginning of the execution. A straightforward way of avoiding such possibly unbounded rollback propagation is to perform eager checkpoint coordination as shown in Fig. 3(b) where $b_{i,x}$ denotes the xth basic checkpoint of p_i. Whenever a process takes a basic checkpoint, *coordination messages* (dotted lines) are broadcast to request the cooperation in making a consistent set of checkpoints [19]. Let B be the total number of basic checkpoints and I be the total number of induced checkpoints. We define the *induction ratio* as

$$\mathcal{R} = \frac{I}{B}$$

which is a measure of the overhead for performing communication-induced checkpoint coordination. Clearly, eager checkpoint coordination has $\mathcal{R} = N - 1$ and will result in large run-time overhead when N is large. In addition, the $N - 1$ coordination messages per checkpoint session constitute another overhead.

The large overhead of eager checkpoint coordination results from its pessimistic nature. More specifically, when p_1 in Fig. 3(b) initiates its first basic checkpoint $b_{1,1}$, it "pessimistically" assumes that messages like m_1 will exist in the future and cause $b_{1,1}$ to be inconsistent with its corresponding checkpoint $b_{0,1}$ on p_0. In order to guarantee $b_{1,1}$ belongs to a useful recovery line, p_1 "eagerly" requests p_0's cooperation at the time $b_{1,1}$ is initiated. In contrast, lazy checkpoint coordination adopts an optimistic approach by assuming that $b_{0,1}$ will be consistent with $b_{1,1}$. If the assumption turns out to be true, no explicit coordination is necessary. An extra checkpoint will be induced on p_0 only when message m_1 indicates that the assumption has failed (Fig. 3(c)). From another point of view, such a scheme "lazily" delays the broadcast of the coordination messages and implicitly piggybacks them on future normal messages [21]. Both checkpoint and message overhead can therefore be reduced.

However, given a basic checkpoint pattern, the number of induced checkpoints in the above scheme is determined by the communication pattern and is not otherwise controllable. In the worst case, the induction ratio $\mathcal{R}$ can still be $N - 1$ as illustrated in Fig. 3(c). In order to further reduce the overhead, we can perform even "lazier" coordination by only enforcing the consistency between checkpoints $c_{0,nZ}$ and $c_{1,nZ}$ where Z is again the *laziness* and n is an integer. Fig. 3(d) shows the case of $Z = 2$. No checkpoint is induced until the message m_2 indicates the inconsistency between $b_{1,2}$ and $b_{0,2}$. The number of induced checkpoints is then reduced from 8 to 2 at the cost of potentially larger rollback distance.

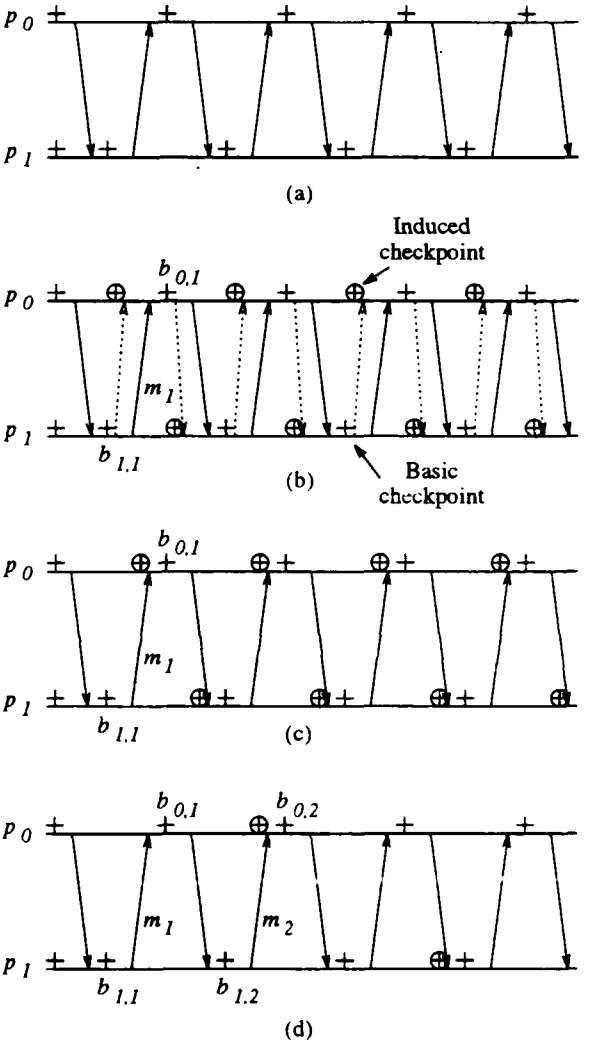


Figure 3: Communication-induced checkpoint coordination. (a) checkpoint and communication pattern; (b) eager checkpoint coordination; (c) lazy checkpoint coordination with laziness = 1; (d) lazy checkpoint coordination with laziness = 2.

3.2 The Protocol

Our approach is to incorporate lazy checkpoint coordination into the uncoordinated checkpointing scheme as a mechanism for bounding rollback propagation. Therefore, the checkpointing and recovery protocol can be built on top of the one described in Section 2. The laziness Z is a predetermined system parameter known to all processes. During normal execution, each process p_i maintains a variable V which is initialized to be Z and incremented by Z each time $c_{i,nZ}$ is taken. When p_i at its xth checkpoint interval is about to process a message m tagged with the sender p_j's checkpoint interval number $y \geq V$, p_i is forced to take the checkpoint $c_{i,lZ}$ where $l = \lfloor y/Z \rfloor$. In other words, if m was sent after $c_{j,lZ}$ had been taken, it must be processed by p_i after $c_{i,lZ}$ is induced. Notice that all the checkpoints $c_{i,w}$ with $x < w < lZ$ become *dummy checkpoints* which overlap with $c_{i,lZ}$.

In addition to the centralized garbage collection procedure as described in Section 2, a simple distributed algorithm can also be used for low-cost garbage collection. The basic idea is that if the current checkpoint interval number of every process has exceeded nZ, all the checkpoints $c_{j,y}$ with $y < nZ$ become obsolete with respect to the consistent set of checkpoints $\{c_{i,nZ} : 0 \leq i \leq N-1\}$ and therefore can be discarded. Each process p_j needs to maintain a *checkpointing progress vector CP_progress*$[N]$ which records the highest checkpoint interval number of every process known to p_j based on the information included in each message. More efficient garbage collection can be achieved by periodically piggybacking the *CP_progress*$[N]$ vector on normal messages.

Although $\{c_{i,nZ} : 0 \leq i \leq N-1\}$ always forms a consistent set of checkpoints, the two-phase recovery procedure described in Section 2 should still be used to search for the local recovery line in order to minimize the number of rolled-back processes and the rollback distances. One possible optimization is that the dependency information associated with the garbage checkpoints determined locally based on *CP_progress*$[N]$ needs not be collected, thus reducing the size of the checkpoint graph.

4 Overhead Analysis

Since the checkpoint overhead of the lazy checkpoint coordination scheme depends on the run-time dynamic communication pattern, it is important to analyze and estimate the potential extra overhead resulting from the induced checkpoints. We will first show that, without any constraints on the relative checkpointing progress of each process, the worst-case induction ratio is $(N-1)/Z$. While under certain conditions which are typically met by real applications, the upper bound on the induction ratio can be shown to be independent of N.

4.1 Worst-Case Analysis

Our approach to worst-case analysis consists of two steps. First, given any basic checkpoint pattern, we construct the worst-case communication pattern. Secondly, given any system with N processes and laziness Z, we derive the worst-case induction ratio as a function of N and Z by considering these worst-case communication patterns.

For the purpose of presentation, we assume every checkpoint $c^{\mathcal{P}}_{i,x}$ in a checkpoint and communication pattern $\mathcal{P}$ is associated with a global time stamp $t(c^{\mathcal{P}}_{i,x})$. For any n, define $c^{\mathcal{P}}_{*,nZ} = c^{\mathcal{P}}_{i,nZ}$ if $t(c^{\mathcal{P}}_{i,nZ}) \leq t(c^{\mathcal{P}}_{j,nZ})$ for all $0 \leq j \leq N-1$, i.e., $c^{\mathcal{P}}_{*,nZ}$ denotes *the earliest checkpoint #nZ among all processes*. Given any basic checkpoint pattern and laziness Z, we construct the communication pattern $\mathcal{P}_0$ as follows.[3] If $c^{\mathcal{P}_0}_{*,nZ} = c^{\mathcal{P}_0}_{i,nZ}$, then p_i sends a message to every other process p_j and induces $c^{\mathcal{P}_0}_{j,nZ}$ with $t(c^{\mathcal{P}_0}_{j,nZ}) \approx t(c^{\mathcal{P}_0}_{i,nZ})$. Fig. 4(a) shows an example of $\mathcal{P}_0$ with $Z = 2$. We will call the interval between $t(c^{\mathcal{P}_0}_{*,(n-1)Z})$ and $t(c^{\mathcal{P}_0}_{*,nZ})$ the *induction session #n* which includes all the induced checkpoints $c^{\mathcal{P}_0}_{j,nZ}$.

Since the induction of any checkpoint $c^{\mathcal{P}}_{j,nZ}$ (and hence any possible dummy checkpoints $c^{\mathcal{P}}_{j,y}$, $(n-1)Z < y < nZ$) cannot happen until the first checkpoint #nZ, say $c^{\mathcal{P}}_{i,nZ}$, is taken, p_i needs to take Z consecutive basic checkpoints by itself in order to reach $c^{\mathcal{P}}_{i,nZ}$, as stated in Property 1.

PROPERTY 1 *If $c^{\mathcal{P}}_{*,nZ} = c^{\mathcal{P}}_{i,nZ}$, then the Z checkpoints $c^{\mathcal{P}}_{i,x}$, $(n-1)Z < x \leq Z$, must be basic checkpoints.*

By the construction of $\mathcal{P}_0$, it is not hard to see that, for any n, $\mathcal{P}_0$ always has the earliest $c^{\mathcal{P}}_{*,nZ}$ among all communication patterns, given the basic checkpoint pattern. (Formal proofs can be found in the complete technical report [30].) Hence, $\mathcal{P}_0$ must possess the largest number of $c^{\mathcal{P}}_{*,nZ}$'s. Since each $c^{\mathcal{P}}_{*,nZ}$ in $\mathcal{P}_0$ also induces the largest possible number ($N-1$) of induced checkpoints, the total number of induced checkpoints in $\mathcal{P}_0$ must be the largest and so we have the following property.

PROPERTY 2 *Given a basic checkpoint pattern, $\mathcal{P}_0$ is the worst-case communication pattern resulting in the largest induction ratio.*

Property 2 states that, for the analysis of worst-case induction ratio, we only need to consider the communication

[3] When it is clear from the context that the basic checkpoint pattern is fixed, the same notation for the *checkpoint and communication pattern* will also be used to refer to the *communication pattern*.

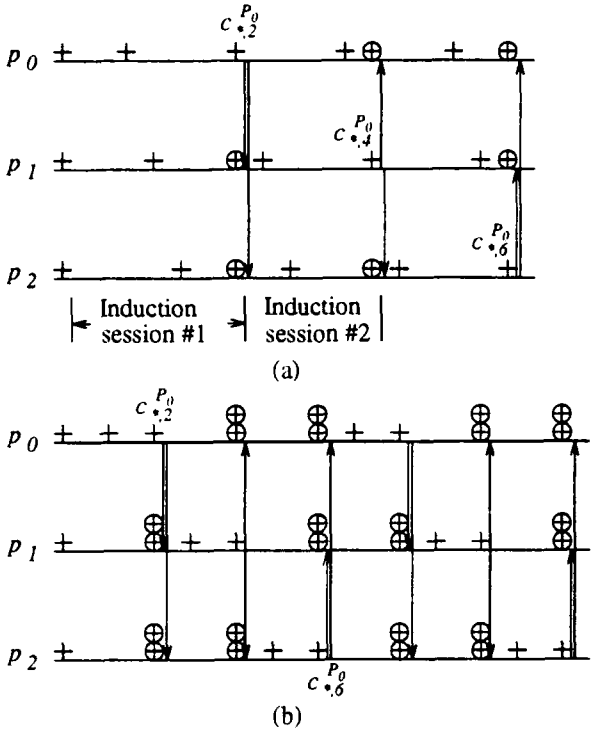


Figure 4: (a) Worst-case communication pattern (b) worst-case checkpoint and communication pattern.

pattern $\mathcal{P}_0$ for each basic checkpoint pattern. Since every $\mathcal{P}_0$ has well-defined induction sessions as shown in Fig. 4, the derivations can be greatly simplified.

From Property 1, at least Z basic checkpoints are needed to induce at most $N-1$ checkpoints and so we have an upper bound on the induction ratio

$$\mathcal{R} \leq \frac{N-1}{Z} \tag{1}$$

It is also the worst-case induction ratio achievable by some $\mathcal{P}_0$ for which an example with $Z = 2$ and $N = 3$ is shown in Fig. 4(b). (The stacked checkpoints indicate that each dummy checkpoint $c^{\mathcal{P}_0}_{i,2n-1}$ overlaps with the induced checkpoint $c^{\mathcal{P}_0}_{i,2n}$.)

4.2 The Upper Bound under Constraints

The upper bound in Eq. (1) was derived under no constraints on the checkpoint and communication pattern. Since it is of order $\mathbf{O}(N)$, the induction ratio may be unacceptably high for systems with a large number of processes. However, a closer look at the two patterns in Fig. 4 reveals that the situation in (b) which results in the worst-case induction ratio is less likely to happen for applications where the basic checkpoint intervals typically do not vary too much. For example in (b), it is very likely for p_0 to take at least one basic checkpoint between $t(c^{\mathcal{P}_0}_{*,2})$ and $t(c^{\mathcal{P}_0}_{*,6})$. We will show that under the following constraints which are satisfied in many applications, the upper bound on the induction ratio is independent of N for $Z \geq 2$. (For the case of $Z = 1$, Fig. 3(c) demonstrates that the worst-case induction ratio of $(N-1)/Z = N-1$ is always achievable and cannot be reduced.)

Constraint-1: Let Q denote *the maximum ratio of any two basic checkpoint intervals.* Although each process is allowed to take its basic checkpoints at its own pace, Q is typically bounded by a small constant $\hat{Q}$. (For example, $\hat{Q}$ is 2 or 3 for our experiments described in the next section.)

Constraint-2: Let L be the number of *complete* induction sessions in $\mathcal{P}_0$. The applications employing checkpointing and rollback recovery are usually long-running programs, which implies $Z \cdot L$ is quite large. In particular, we assume $Z \cdot L \gg \lceil Q \rceil$.

From Property 1, each induction session must contain Z consecutive basic checkpoints and hence at least $Z-1$ basic checkpoint intervals. Let S denote the following set of integers

$$S = \{m : m \cdot (Z-1) \geq Q \text{ and } m \leq \lceil Q \rceil\}.$$

For $Z \geq 2$, S contains at least one element, namely, $\lceil Q \rceil$. Let M be *the minimum element* of S. We define an *M-session* as consisting of M consecutive induction sessions, session $\#((n-1)M+1)$ through session $\#nM$. Our approach is based on the observation that within an M-session, every process either takes at least one set of Z consecutive basic checkpoints which defines one of the induction sessions, or takes at least one basic checkpoint due to Constraint-1. Since, within an M-session, the number of induced checkpoints is $M \cdot (N-1)$ and the number of basic checkpoints is at least N, the upper bound on the induction ratio is independent of N.

THEOREM 1 *Under the above two constraints, the induction ratio $\mathcal{R} < \lceil Q \rceil$ for laziness $Z \geq 2$, where Q is the maximum ratio of any two basic checkpoint intervals.*

Proof. Again we only have to consider $\mathcal{P}_0$ for each basic checkpoint pattern. There are $L_M = \lfloor L/M \rfloor$ complete M-sessions, each containing $M \cdot (N-1)$ induced checkpoints. We distinguish the following two cases.
(a) $N < M$: From Eq. (1), $\mathcal{R} \leq \frac{N-1}{Z} < N < M \leq \lceil Q \rceil$.
(b) $N \geq M$: First we consider the number of induced checkpoints I. If $Z \geq Q+1$, then $M = 1$ and $I = L \cdot (N-1)$. If $Z < Q+1$, $Z \cdot L \gg \lceil Q \rceil$ in Constraint 2

implies $L \gg \lceil Q \rceil$. Since $M \leq \lceil Q \rceil$, we have $L/M \gg 1$; so $L_M \gg 1$ and $I \approx L_M \cdot M \cdot (N-1)$. In either case, $I \approx L_M \cdot M \cdot (N-1)$.

Now consider the number of basic checkpoints B. For each induction session #n, the process p_i with $c_{i,nZ}^{P_0} = c_{*,nZ}^{P_0}$ must contribute Z basic checkpoints and therefore the length of each induction session is at least $Z-1$ basic checkpoint intervals. Within each M-session, at least $N-M$ processes do not contain $c_{*,nZ}^{P_0}$ for any n. By the definition of Q, these $N-M$ processes must each contribute at least $\lfloor \frac{M \cdot (Z-1)}{Q} \rfloor$ basic checkpoints. Therefore,

$$B \geq L_M \cdot (M \cdot Z + (N-M) \cdot \lfloor \frac{M \cdot (Z-1)}{Q} \rfloor) \text{ and}$$

$$\mathcal{R} = \frac{I}{B} \leq \frac{M \cdot (N-1)}{M \cdot Z + (N-M) \cdot \lfloor \frac{M \cdot (Z-1)}{Q} \rfloor}. \quad (2)$$

Since $Z > 1$ and $\frac{M \cdot (Z-1)}{Q} \geq 1$ by definition, we have

$$\mathcal{R} < \frac{M \cdot (N-1)}{M + (N-M)} < M \leq \lceil Q \rceil. \quad (3)$$

as required. □

Combining Eq. (1) (for $Z = 1$ and Case (a)) and Eq. (2), we then define the refined upper bound, called the Q-bound, as follows

$$Q\text{-bound} = \frac{M \cdot (N-1)}{M \cdot Z + [N \geq M] \cdot ((N-M) \cdot \lfloor \frac{M \cdot (Z-1)}{Q} \rfloor)}$$

where $[N \geq M] = 1$ if $N \geq M$ is true and 0 otherwise.

5 Experimental Results

Four parallel programs written in the *Chare Kernel* language [31] are used for the communication trace-driven simulation. The Chare Kernel has been developed as a machine-independent *message-driven* parallel language. Program traces used in this paper are collected from an Encore Multimax 510.

The four programs include two computer-aided circuit design applications, Test Generation and Logic Synthesis, and two search applications, Knight Tour and N-Queen. The execution times are between 25 and 45 minutes (see Table 1). The predetermined minimum basic checkpoint interval is chosen to be 2 minutes. A variable *Next_CP_Time* is initialized to 2 minutes. Each process checks its local clock after processing every 100 messages. If the clock time exceeds *Next_CP_Time*, a basic checkpoint is inserted and *Next_CP_Time* is incremented by 2 minutes. The resulting average basic checkpoint interval (CPI) for each program is listed in Table 1. Before processing a new message, each process also checks if it needs to take an induced checkpoint, as described in Section 3. All reported numbers are averaged over five runs.

We expect the variation of the basic checkpoint interval to be small because of the way it is maintained. In particular, we choose $Q = 2$ to estimate the induction ratio. The exact value of Q for each program is listed in Table 1. Although Q is slightly greater than 2 for the first two programs, the numbers listed in the row of "Under-2 percentage" show that a very high percentage of the basic checkpoint intervals are covered by $Q = 2$ which thus serves as a good approximation. Fig. 5 plots the Q-bounds against the worst-case and the actual induction ratios for the four programs. It demonstrates that the Q-bound provides a good estimate of the induction ratio. The large difference in the ratio between $Z = 1$ and $Z \geq 2$ confirms that our generalization of the idea of communication-induced checkpoint coordination as described in [25] can significantly reduce the extra checkpoint overhead.

Fig. 6 plots the average rollback distances in terms of the number of average basic CPIs for the four programs. We use 0.5 for $Z = 1$ and $(Z-1)/2$ for $Z \geq 2$ in the "Estimated" curve. Figs. 5 and 6 illustrate that lazy checkpoint coordination provides a flexible trade-off between coordination overhead and recovery efficiency.

6 Summary

We have proposed the technique of lazy checkpoint coordination and incorporated it into an uncoordinated checkpointing protocol as a mechanism for bounding rollback propagation. Recovery line progression is guaranteed by performing communication-induced checkpoint coordination only when the predetermined consistency criterion is about to be violated. The notion of laziness has been introduced to provide a trade-off between extra checkpoints during normal execution versus the average rollback distance for recovery. Overhead analysis shows that the upper bound on the induction ratio, i.e., the number of induced checkpoints divided by the number of basic checkpoints, is related to the maximum ratio between the basic checkpoint intervals. Communication trace-driven simulation results for four parallel programs showed that our analysis can provide a good estimate of the induction ratio, and that lazy checkpoint coordination can significantly reduce the number of induced checkpoints.

Acknowledgement

The authors wish to express their sincere thanks to the anonymous referees for their valuable comments, to B. Ramkumar, K. De and P. Banerjee for their parallel programs and to L. V. Kalé for access to the Chare Kernel.

Table 1: Execution and checkpoint parameters of the parallel programs.

Programs	Test Generation	Logic Synthesis	Knight Tour	N-Queen
Number of processors	8	6	8	6
Execution time (sec)	2,076	1,736	2,436	1,567
Number of messages	28,219	411,733	104,170	25,880
Average basic CPI (sec)	158	140	132	139
Q	2.17	2.48	1.42	1.55
Under-2 percentage	99.6%	97.0%	100%	100%

References

[1] K. Tsuruoka, A. Kaneko, and Y. Nishihara, "Dynamic recovery schemes for distributed processes," in *Proc. IEEE 2nd Symp. on Reliability in Distributed Software and Database Systems*, pp. 124–130, 1981.

[2] B. Bhargava and S. R. Lian, "Independent checkpointing and concurrent rollback for recovery - An optimistic approach," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 3–12, 1988.

[3] Y. M. Wang and W. K. Fuchs, "Optimistic message logging for independent checkpointing in message-passing systems," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 147–154, Oct. 1992.

[4] B. Randell, "System structure for software fault tolerance," *IEEE Trans. on Software Engineering*, Vol. SE-1, No. 2, pp. 220–232, June 1975.

[5] Y. M. Wang and W. K. Fuchs, "Scheduling message processing for reducing rollback propagation," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 204–211, July 1992.

[6] K. H. Kim, J. H. You, and A. Abouelnaga, "A scheme for coordinated execution of independently designed recoverable distributed processes," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 130–135, 1986.

[7] K. Venkatesh, T. Radhakrishnan, and H. F. Li, "Optimal checkpointing and local recording for domino-free rollback recovery," *Information Processing Letters*, Vol. 25, pp. 295–303, July 1987.

[8] D. L. Russel, "State restoration in systems of communicating processes," *IEEE Trans. on Software Engineering*, Vol. SE-6, No. 2, pp. 183–194, Mar. 1980.

[9] A. Borg, J. Baumbach, and S. Glazer, "A message system supporting fault-tolerance," in *Proc. 9th ACM Symp. on Operating Systems Principles*, pp. 90–99, 1983.

[10] M. L. Powell and D. L. Presotto, "Publishing: A reliable broadcast communication mechanism," in *Proc. 9th ACM Symp. on Operating Systems Principles*, pp. 100–109, 1983.

[11] R. E. Strom and S. Yemini, "Optimistic recovery in distributed systems," *ACM Trans. on Computer Systems*, Vol. 3, No. 3, pp. 204–226, Aug. 1985.

[12] R. E. Strom, D. F. Bacon, and S. A. Yemini, "Volatile logging in n-fault-tolerant distributed systems," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 44–49, 1988.

[13] A. P. Sistla and J. L. Welch, "Efficient distributed recovery using message logging," in *Proc. 8th ACM Symposium on Principles of Distributed Computing*, pp. 223–238, 1989.

[14] D. B. Johnson and W. Zwaenepoel, "Recovery in distributed systems using optimistic message logging and checkpointing," *J. of Algorithms*, Vol. 11, pp. 462–491, 1990.

[15] T. T.-Y. Juang and S. Venkatesan, "Crash recovery with little overhead," in *Proc. IEEE Int'l Conf. on Distributed Computing Systems*, pp. 454–461, 1991.

[16] E. N. Elnozahy and W. Zwaenepoel, "Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit," *IEEE Trans. on Computers*, Vol. 41, No. 5, pp. 526–531, May 1992.

[17] B. H. L. Alvisi and K. Marzullo, "Nonblocking and orphan-free message logging protocols," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 145–154, 1993.

[18] Y. M. Wang, Y. Huang, and W. K. Fuchs, "Progressive retry for software error recovery in distributed systems," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 138–144, June 1993.

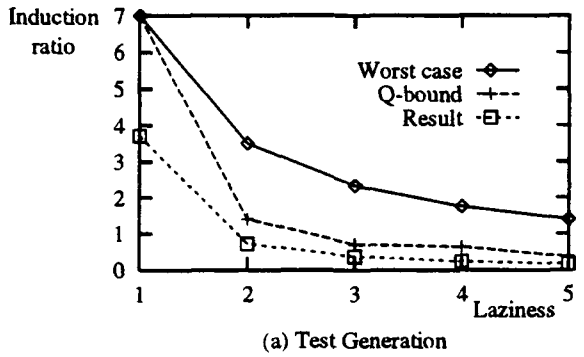
[19] K. G. Shin and Y.-H. Lee, "Evaluation of error recovery blocks used for cooperating processes," *IEEE Trans. on Software Engineering*, Vol. 10, No. 6, No. 6, pp. 692–700, 1984.

[20] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. on Computer Systems*, Vol. 3, No. 1, pp. 63–75, Feb. 1985.

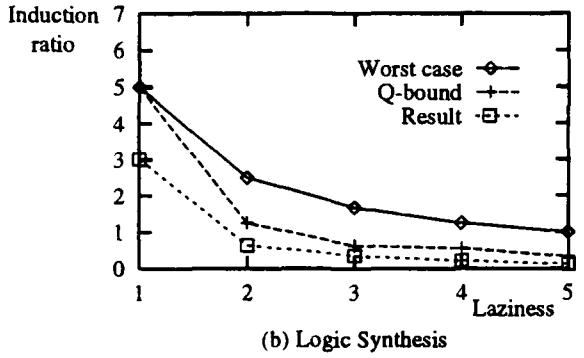
[21] T. H. Lai and T. H. Yang, "On distributed snapshots," *Information Processing Letters*, Vol. 25, pp. 153–158, May 1987.

[22] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. on Software Engineering*, Vol. SE-13, No. 1, pp. 23–31, Jan. 1987.

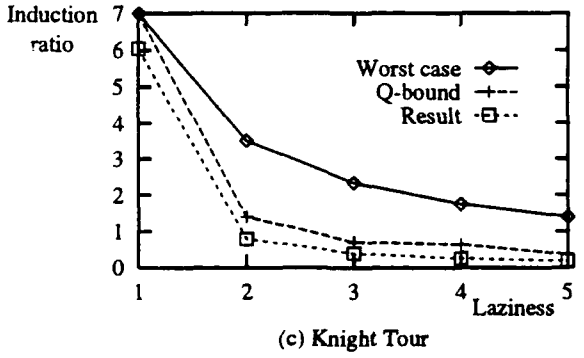
[23] K. Li, J. F. Naughton, and J. S. Plank, "Checkpointing multicomputer applications," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 2–11, 1991.



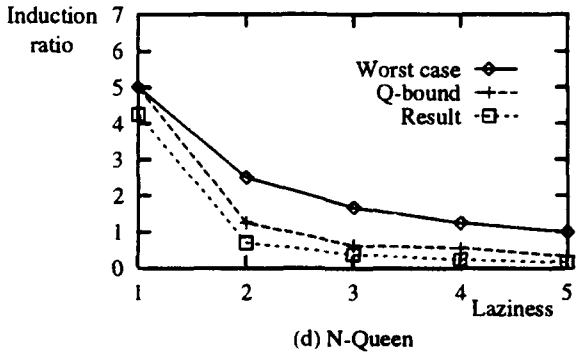
(a) Test Generation



(b) Logic Synthesis



(c) Knight Tour



(d) N-Queen

Figure 5: Checkpoint coordination overhead (induction ratio) as a function of laziness.

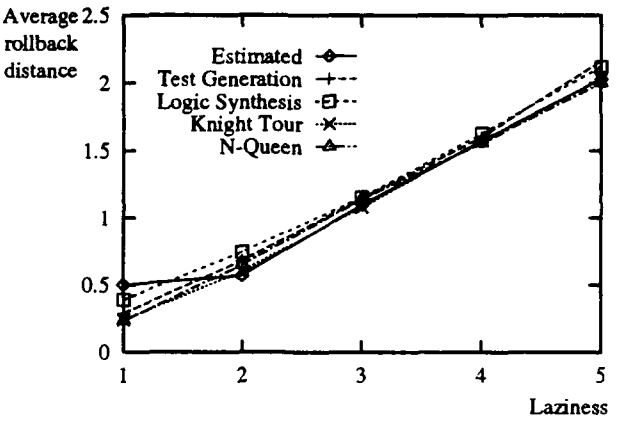


Figure 6: Average rollback distance (number of average basic CPIs) as a function of laziness.

[24] M. F. Kaashoek, R. Michiels, H. E. Bal, and A. S. Tanenbaum, "Transparent fault-tolerance in parallel Orca programs," Tech. Rep. IR-258, Vrije Universiteit, Amsterdam, Oct. 1991.

[25] D. Briatico, A. Ciuffoletti, and L. Simoncini, "A distributed domino-effect free recovery algorithm," in *Proc. IEEE 4th Symp. on Reliability in Distributed Software and Database Systems*, pp. 207–215, 1984.

[26] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems," *ACM Trans. on Computer Systems*, Vol. 1, No. 3, pp. 222–238, Aug. 1983.

[27] A. Borg, W. Blau, W. Graetsch, F. Herrmann, and W. Oberle, "Fault tolerance under UNIX," *ACM Trans. on Computer Systems*, Vol. 7, No. 1, pp. 1–24, Feb. 1989.

[28] Y. M. Wang, A. Lowry, and W. K. Fuchs, "Consistent global checkpoints based on direct dependency tracking," Research Report RC 18465, IBM T.J. Watson Research Center, Yorktown Heights, New York, Oct. 1992.

[29] Y. M. Wang, P. Y. Chung, I. J. Lin, and W. K. Fuchs, "Checkpoint space reclamation for independent checkpointing in message-passing systems," Tech. Rep. CRHC-92-06, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1992.

[30] Y. M. Wang and W. K. Fuchs, "Lazy checkpoint coordination for bounding rollback propagation," Tech. Rep. CRHC-92-26, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1992.

[31] W. Shu and L. V. Kalé, "Chare kernel - A runtime support system for parallel computations," *J. Parallel and Distributed Computing*, Vol. 11, pp. 198–211, 1991.