November 1993

UILU-ENG-93-2251
CRHC-93-24

Center for Reliable and High-Performance Computing

SPACE RECLAMATION FOR UNCOORDINATED CHECKPOINTING IN MESSAGE-PASSING SYSTEMS

Yi-Min Wang

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-93-2251 CRHC-93-24	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois	6b. OFFICE SYMBOL (If applicable) N/A
6c. ADDRESS (City, State, and ZIP Code) 1301 W. Main St. Urbana, IL 61801	7a. NAME OF MONITORING ORGANIZATION NASA 7b. ADDRESS (City, State, and ZIP Code) Moffett Field, CA 94035
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 7a	8b. OFFICE SYMBOL (If applicable)
8c. ADDRESS (City, State, and ZIP Code) 7b	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NASA NAG 1-613 10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Space Reclamation for Uncoordinated Checkpointing in Message-Passing Systems

12. PERSONAL AUTHOR(S) WANG, Yi-Min

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	93-11-19	116

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	rollback, fault tolerance, parallel and distributed systems, message logging, garbage collection, independent checkpointi

19. ABSTRACT

Checkpointing and rollback recovery are techniques that can provide efficient recovery from transient process failures. In a message-passing system, the rollback of a message sender may cause the rollback of the corresponding receiver, and the system needs to roll back to a consistent set of checkpoints called recovery line. If the processes are allowed to take uncoordinated checkpoints, the above rollback propagation may result in the domino effect which prevents recovery line progression. Traditionally, only obsolete checkpoints before the global recovery line can be discarded, and the necessary and sufficient condition for identifying all garbage checkpoints has remained an open problem.

We derive a necessary and sufficient condition for achieving optimal garbage collection, and we prove that the number of useful checkpoints is in fact bounded by $N(N+1)/2$ where N is the number of processes. Our approach is based on the maximum-sized antichain model of consistent global checkpoints and the technique of recovery line transformation and decomposition. We also show that, for systems requiring message logging to record in-transit messages, the same approach can be used to achieve optimal message log reclamation. As a final topic, we describe a unifying framework by considering checkpoint coordination and exploiting piecewise determinism as mechanisms for bounding rollback propagation, and demonstrate the applicability of the optimal garbage collection algorithm to domino-freee recovery protocols.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

SPACE RECLAMATION FOR UNCOORDINATED CHECKPOINTING IN MESSAGE-PASSING SYSTEMS

BY

YI-MIN WANG

B.S., National Taiwan University, 1986
M.S., University of Illinois at Urbana-Champaign, 1990

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1993

Urbana, Illinois

SPACE RECLAMATION FOR UNCOORDINATED CHECKPOINTING IN MESSAGE-PASSING SYSTEMS

Yi-Min Wang, Ph.D.
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, 1993
W. Kent Fuchs, Advisor

Checkpointing and rollback recovery are techniques that can provide efficient recovery from transient process failures. In a message-passing system, the rollback of a message sender may cause the rollback of the corresponding receiver, and the system needs to roll back to a consistent set of checkpoints called the recovery line. If the processes are allowed to take uncoordinated checkpoints, the above rollback propagation may result in the domino effect which prevents recovery line progression. Traditionally, only obsolete checkpoints before the global recovery line can be discarded, and the necessary and sufficient condition for identifying all garbage checkpoints has remained an open problem.

In this thesis, we derive a necessary and sufficient condition for achieving optimal garbage collection, and we prove that the number of useful checkpoints is in fact bounded by $N(N+1)/2$ where N is the number of processes. Our approach is based on the maximum-sized antichain model of consistent global checkpoints and the technique of recovery line transformation and decomposition. We also show that, for systems requiring message logging to record in-transit messages, the same approach can be used to achieve optimal message log reclamation. As a final topic, we describe a unifying framework by considering checkpoint coordination and exploiting piecewise determinism as mechanisms

for bounding rollback propagation, and demonstrate the applicability of the optimal garbage collection algorithm to domino-free recovery protocols.

DEDICATION

Dedicated to Pi-Yu, Jeffrey, and Andrew.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis advisor, Professor W. Kent Fuchs, for his support and guidance, and so many things I have learned from him throughout this thesis research.

I would also like to thank Professors Prith Banerjee, Geneva Belford, Ravi Iyer and Michael Loui for serving on my committee, and all my colleagues in the Coordinated Science Laboratory for their friendship and assistance. I wish to express my thanks to Professor Michael Loui for coaching my presentations, to Andy Lowry (IBM), Yennun Huang (AT&T), In-Jen Lin, Shyh-Kwei Chen and Weiping Shi for their stimulating discussions, and to Jungsheng Long, Balkrishna Ramkumar and Kaushik De for their help with the experimental results.

Last, but not least, I would like to thank my lovely wife, Pi-Yu, for being my best research partner, my parents for their support, understanding and encouragement, and my sons, Jeffrey and Andrew, for coming into the world in time to contribute so much inspiration to this dissertation.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

1. INTRODUCTION

1.1 Checkpointing and Rollback Recovery

Checkpointing and rollback recovery provide for recovery from transient process failures. During normal execution, the state of each process is periodically saved on stable storage as a *checkpoint*. When a failure occurs, the process can roll back to a previous checkpoint by reloading the checkpointed state to avoid costly reexecution from the very beginning. In a message-passing system, *rollback propagation* can occur when the rollback of a message sender results in the rollback of the corresponding receiver. The system is then required to roll back to the latest available consistent set of checkpoints called the *recovery line* to ensure correct recovery with a minimum amount of rollback. In the worst case, cascading rollback propagation [1] may result in the *domino effect* [2, 3] which prevents recovery line progression.

Numerous checkpointing and recovery techniques for message-passing systems have been proposed in the literature. They can be classified into three primary categories: uncoordinated checkpointing, coordinated checkpointing and the log-based approach. **Uncoordinated checkpointing** [4–6] allows each process to take its checkpoints independently, without coordinating with any other processes. It allows maximum process autonomy and general nondeterministic executions, but suffers from potential domino

effects and the large space overhead for maintaining multiple checkpoints of each process. Processes are allowed to take uncoordinated checkpoints, and the dependencies among the checkpoints caused by message communication are recorded through *dependency tracking*. The recovery line is unknown during normal execution and is computed at the time of recovery based on the dependency information. Rollback propagation can be eliminated by taking a checkpoint immediately after sending every message [7], and domino-free recovery can be achieved by inserting a checkpoint before processing any message carrying a new dependency [8, 9], or by inserting a checkpoint between every pair of consecutive **send** and **receive** events (in that order) [1].

Coordinated checkpointing eliminates the domino effect by sacrificing a certain degree of process autonomy and incurring run-time and message overhead. Usually, whenever a process takes a checkpoint, it broadcasts a *coordination message* to force all of the other processes to take appropriate checkpoints to guarantee that the resulting set of checkpoints is consistent [10–18]. The number of processes required to participate in each checkpointing session can be reduced by monitoring the recent message exchanges [19]. The extra message overhead can be avoided by piggybacking the coordination messages on subsequent normal messages [20–22], or by taking advantage of the existing clock synchronization mechanisms [23–25].

The **log-based approach** assumes the *piecewise deterministic execution model* [26] which views process execution as consisting of a number of deterministic *state intervals*,

each started by a nondeterministic event such as processing a new message. Nondeterministic event logging, in addition to checkpointing, is employed to reduce rollback propagation through deterministic state reconstruction. *Synchronous message logging* protocols [27–29] log each message upon receipt. Since the process state from which any message is sent can always be reconstructed through message replaying, rollback propagation is completely eliminated. *Asynchronous message logging* protocols [26, 30–41] reduce logging overhead by grouping several messages in a single write operation to stable storage. Although rollback propagation may occur when not-yet-logged messages are lost upon a failure, recovery line progression is guaranteed as long as every message is eventually logged [26, 33].

The main focus of this thesis is on uncoordinated checkpointing and, in particular, the garbage collection procedure for reclaiming the storage space of those checkpoints that are no longer useful. Traditionally, garbage collection for uncoordinated checkpointing has been based on the notion of *obsolete checkpoints*: the global recovery line which suffices to recover from the failure of the entire system is computed; then all of the obsolete checkpoints before that recovery line are no longer useful and can be discarded. In contrast, all of the nonobsolete checkpoints have been assumed to be possibly useful for some future recovery and should be retained. With the possibility of domino effects, the number of nonobsolete checkpoints is potentially unbounded.

Motivated by the observation that being obsolete is simply a sufficient condition for being garbage, we derive a necessary and sufficient condition for identifying all garbage

checkpoints, which leads to an optimal garbage collection algorithm and the lowest upper bound on the number of nongarbage checkpoints. Our approach is to model consistent global checkpoints as maximum-sized antichains of the partially ordered set generated by the *happened before* relation between the checkpoints. We define a recovery line transformation and decomposition, and we demonstrate that any nongarbage checkpoint belonging to a possible future recovery line must also be contained in one of the N "immediate future" recovery lines, where N is the number of processes. It is also shown that these N recovery lines can contain at most $N(N+1)/2$ distinct nongarbage checkpoints.

Usually, the in-transit messages, i.e., messages "sent but not yet received" with respect to a set of checkpoints, are assumed to be handled by a reliable transmission protocol and do not result in checkpoint inconsistency. We point out that to support the above assumption, the acknowledgement message for every normal message has to be considered as an additional dependency-carrying message which would result in extra rollback propagation. An alternative way of retrieving the in-transit messages is to use message logging. The message logs then constitute another source of space overhead. We demonstrate that the same approach based on recovery line transformation and decomposition can be used to develop an optimal message log reclamation algorithm. More specifically, we show that any message that can possibly become an in-transit message in the future must also be an in-transit message with respect to one of the N "immediate future" recovery lines.

Our optimal garbage collection algorithm addresses the space overhead issue of uncoordinated checkpointing, but the possibility of domino effects still remains. In Chapter 3, we extend the applicability of the algorithm to a domino-free unifying framework. Traditionally, uncoordinated checkpointing, coordinated checkpointing, and the log-based approach have been considered three separate approaches, each with its own advantages and disadvantages. The unifying framework provides a different point of view by considering uncoordinated checkpointing as the basic and the most general scheme because it does not require process execution to satisfy the piecewise deterministic model. Checkpoint coordination and message logging for exploiting piecewise determinism are then considered two mechanisms for bounding rollback propagation. We propose a *lazy checkpoint coordination* technique [22] to allow sacrificing a varying degree of process autonomy in exchange for a guarantee of recovery line progression. Message logging whenever piecewise determinism is available is interpreted as placing additional *logical checkpoints* [42] at the end of the state intervals, thereby reducing the rollback distances and hence the possibility of rollback propagation. The unifying framework and the optimal garbage collection algorithm together then provide a flexible, effective, economic way of recovering from transient process failures.

1.2 Checkpoint Consistency

The system considered in this thesis 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 [43], i.e., no corrupted messages can be sent. All processes running on the same *recovery unit* [26] will be rolled back together in response to a failure. For the purpose of presentation, we consider each process to be an individual recovery unit. To allow general nondeterministic execution, we do not assume the piecewise deterministic model. This implies that whenever the sender of a message m rolls back to a point before m was sent to *unsend* m, the corresponding receiver must also roll back to a point before m was processed in order to *unprocess* m.[1] Let $c_{i,x}$ denote the xth checkpoint ($x \geq 0$) of process p_i. Figure 1.1(a) gives such an example. Suppose process p_j rolls back to $c_{j,y}$. Due to the the potential nondeterminism preceding the sending of m, p_j can not guarantee the regeneration of an exact copy of m during its reexecution (even under the fail-stop assumption). Thus, p_i's execution based on the processing of m is no longer valid and p_i should also roll back to nullify the effect of m. The message m which is *unsent* by p_j is called an *orphan message* and results in the *inconsistency* between $c_{j,y}$ and $c_{i,x+1}$. The two checkpoints thus cannot be used together for recovery.

In contrast, Fig. 1.1(b) shows another situation in which message m' is recorded as "sent but not yet received" and hence is called an *in-transit message* with respect to the two checkpoints $c_{i,x}$ and $c_{j,y}$. Suppose that process p_i rolls back to $c_{i,x}$ and *unreceives* m'. A straightforward way of handling such a situation is to also roll back p_j to *unsend* m', a mechanism we call *in-transit message invalidation*. However, such invalidation can

[1] We say a message is *received* by the destination processor and then later *processed* by the destination process. A message results in dependency only after it is processed.

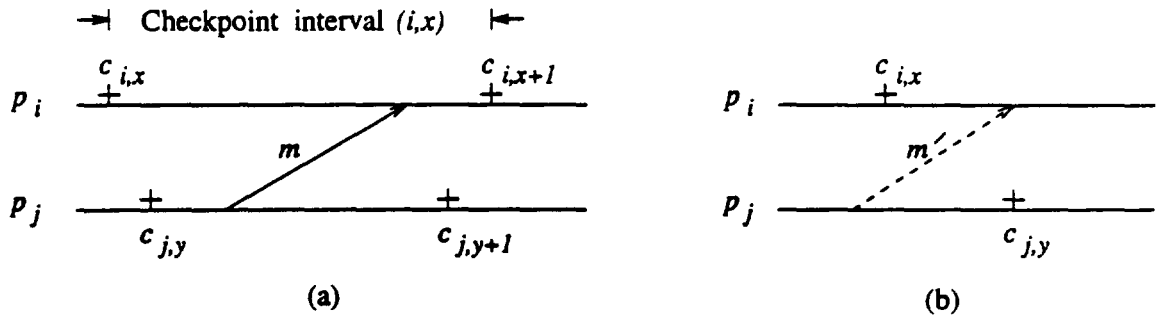


Figure 1.1: Checkpoint consistency (solid line for message processing; dashed line for message receipt). (a) Orphan message m and *inconsistent* checkpoints $c_{i,x+1}$ and $c_{j,y}$; (b) in-transit message m' and *consistent* checkpoints $c_{i,x}$ and $c_{j,y}$.

result in excessive rollback propagation and a higher probability of domino effects. Another commonly used mechanism can be called *in-transit message retrieval.* If during p_i's reexecution from $c_{i,x}$, message m' can be retrieved from a message log or through an end-to-end transmission protocol, then p_i need not request p_j to *unsend* m'. Several approaches employing the above two mechanisms to handle in-transit messages are summarized in the following. The first and the fourth approaches use a combination of invalidation and retrieval; the other two approaches are based completely on the retrieval mechanism.

Approach 1: reliable end-to-end transmission protocol

Koo and Toueg [19] argued that the situation of message m' with respect to the checkpoints $c_{i,x}$ and $c_{j,y}$ in Fig. 1.2(a) is indistinguishable from the situation in which m' is lost in the communication channel during normal execution. Therefore, a reliable end-to-end transmission protocol, which can guarantee the retransmission of any lost

message until it is received by the destination processor, will also be able to retransmit m' after the two processes roll back to $c_{i,x}$ and $c_{j,y}$.

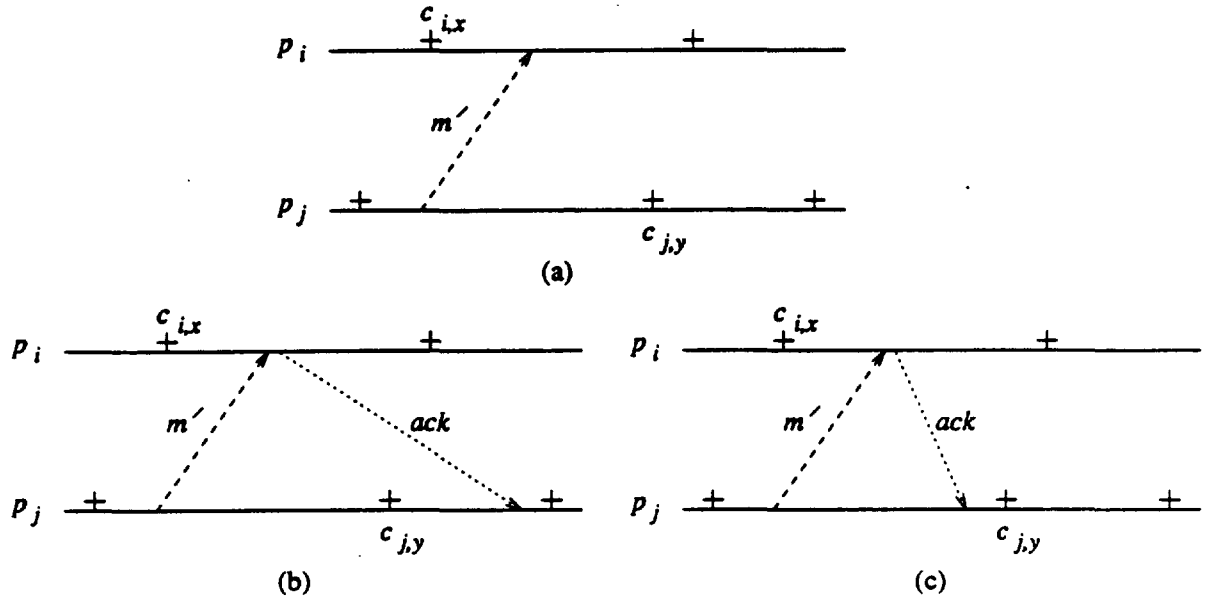


Figure 1.2: In-transit message and checkpoint consistency. (a) In-transit message m'; (b) consistent checkpoints $c_{i,x}$ and $c_{j,y}$; (c) inconsistent checkpoints $c_{i,x}$ and $c_{j,y}$ due to message *ack*.

However, this is true only for the situation shown in Fig. 1.2(b) where $c_{j,y}$ is taken before p_j receives message *ack* (the acknowledge message for m'), and thus will record a copy of message m' as well as the responsibility of retransmitting m' until *ack* comes back. If checkpoint $c_{j,y}$ is taken after p_j receives *ack*, as shown in Fig. 1.2(c), $c_{j,y}$ will simply lose the capability of resending m'. It becomes clear that, to distinguish the two different scenarios, message *ack* has to be treated as an additional dependency-carrying message which can result in extra rollback propagation. More precisely, in Fig. 1.2(b), the rollback of p_i to $c_{i,x}$ requires (through *ack*) that p_j be rolled back to $c_{j,y}$ so that the in-transit message m' can be resent. In Fig. 1.2(c), the rollback of p_i to $c_{i,x}$ causes

(through *ack*) the rollback of p_j to a checkpoint earlier than $c_{j,y}$ in order to *invalidate* m'. We note that the inconsistency between $c_{i,x}$ and $c_{j,y}$ in Fig 1.2(c) is due to the orphan message *ack*, not the in-transit message m'.

Approach 2: synchronous message logging

Treating every acknowledge message as a dependency-carrying message potentially doubles the amount of rollback propagation and makes recovery line progression more difficult. Another way to handle in-transit messages is to use message logging. A *synchronous message logging* protocol logs every incoming message m' upon its arrival and delays the sending of *ack* message until m' is logged. In this way, if the receiver p_i rolls back and *unreceives* m' *before* it logs m', then the sender will resend m' because the corresponding *ack* is never generated; if p_i initiates the rollback *after* it logs m', then p_i can retrieve m' from the log during its reexecution.

Approach 3: asynchronous message logging with sender logging

A synchronous logging protocol logs each incoming message separately and can result in large performance degradation. Asynchronous logging protocols [26] reduce run-time overhead by grouping several messages and logging them later in a single `write` operation. Additional sender logging can be used to maintain a copy of each message which is not yet logged by the receiver and can potentially be lost in the presence of a receiver's failure. Every process keeps the messages it has sent in a volatile log [26] and writes them to stable storage at the next checkpoint. The sender retains the log for each message until it is notified (by a *log_ack* message) that the message has been logged by the receiver. In

this way, every in-transit message can always be retrieved from either the sender's log or the receiver's log. Figure 1.3(a) and (b) illustrate the difference between Approaches 2 and 3 by showing the availability of the message log for m' at different times, as indicated by the shaded bars.

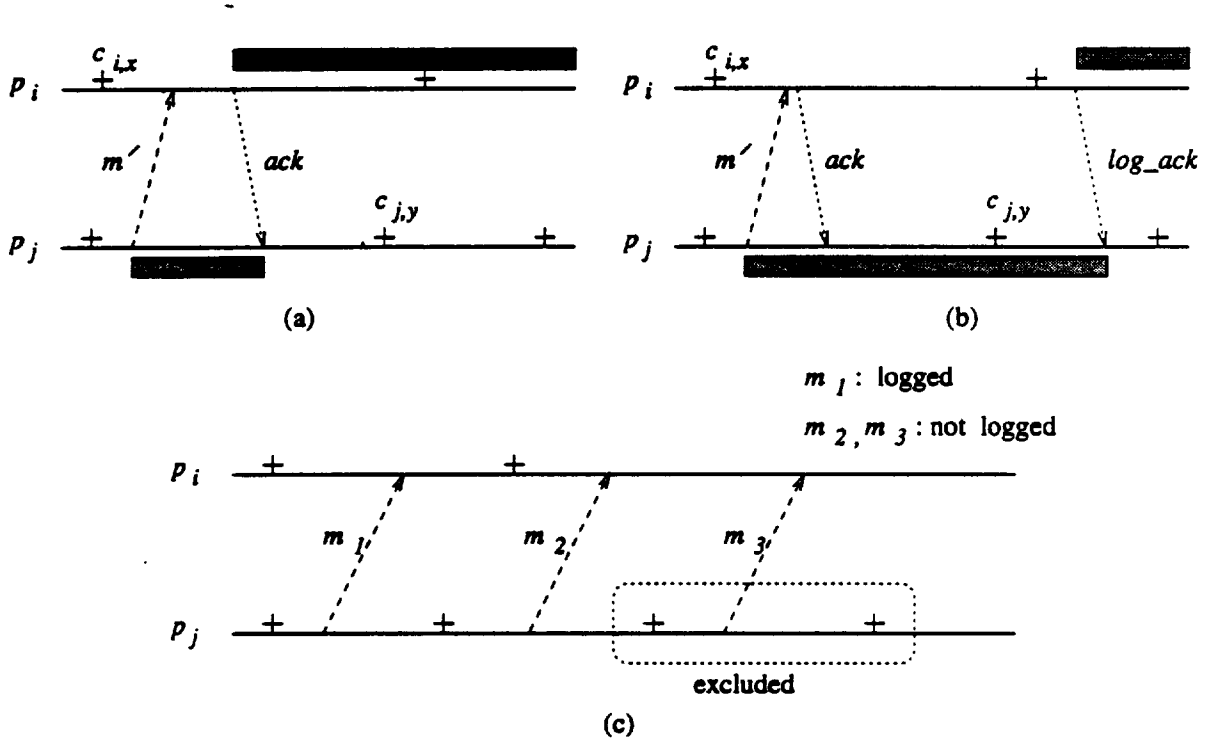


Figure 1.3: Message logging for recording the in-transit messages. (a) Synchronous message logging; (b) asynchronous message logging with sender logging; (c) asynchronous message logging without sender logging. (Shaded bars indicate the availability of the message log for m'.)

Approach 4: asynchronous message logging without sender logging

Without additional sender logging, messages can be lost upon a receiver's failure in an asynchronous logging protocol. One way to remedy such a situation is to compare the set of messages sent with the set of messages logged and consider unavailable those

checkpoints before which there is any "sent but not yet logged" message [6], as illustrated in Fig. 1.3(c). This procedure effectively invalidates all lost messages and ensures that in-transit messages with respect to the computed recovery line can all be retrieved from the receiver's log.

All of the above four approaches can guarantee that messages like m' in Fig. 1.1(b) do not cause the inconsistency between $c_{i,x}$ and $c_{j,y}$. Therefore, the situation shown in Fig. 1.1(a) is the only source of checkpoint inconsistency.

1.3 Uncoordinated Checkpointing Protocol

Having addressed the checkpoint consistency issues, we now describe an uncoordinated checkpointing protocol. Suppose there are N processes in the system. During normal execution, each process takes its *local checkpoints* periodically without coordinating with any other processes. Let (i, x) denote the xth *checkpoint interval* of process p_i between consecutive checkpoints $c_{i,x}$ and $c_{i,x+1}$, as shown in Fig. 1.1(a). Each message is tagged with the current *checkpoint interval number* and the process number of the sender. Each receiver p_i performs *direct dependency tracking* [4, 44] as follows: if a message sent from (j, y) is processed by p_i in (i, x), then the direct dependency of $c_{i,x+1}$ on $c_{j,y}$ is recorded.

A centralized *garbage collection algorithm* can be invoked by any process p_i periodically to reclaim the storage space of garbage checkpoints and possibly message logs (if the in-transit messages are recorded through message logging) that are no longer useful

for any future recovery. First, p_i broadcasts a *request message* to collect the direct dependency information from all other processes. A *checkpoint graph* [4] is constructed, in which each vertex represents a checkpoint and each edge represents a direct dependency (including the implicit dependency of any $c_{j,y+1}$ on $c_{j,y}$). Figure 1.4(b) shows the checkpoint graph corresponding to the checkpoint and communication pattern in (a). The ***rollback propagation algorithm*** listed in Fig. 1.5 is executed on the checkpoint graph to determine the *global recovery line*,[2] which is then broadcast in a *recovery_line message*. All checkpoints and message logs before the global recovery line are *obsolete*, and their space can therefore be reclaimed. Note that processes other than the initiator do not have to block their executions between replying to the *request message* and receiving the *recovery_line message*.

When a process p_j initiates a rollback, it starts a similar two-phase procedure for recovery, except for the following differences. The volatile states of surviving processes remain valid and can be viewed as additional *virtual checkpoints* [5] for constructing an *extended checkpoint graph* of which the recovery line is called a *local recovery line*. Figure. 1.4(c) shows an example in which p_4 initiates a rollback. Every other process is blocked, after supplying p_4 with the dependency information, until it rolls back to the checkpoint as indicated by the local recovery line. Figure. 1.4(d) shows the checkpoint graph immediately after the recovery.

[2]A 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.

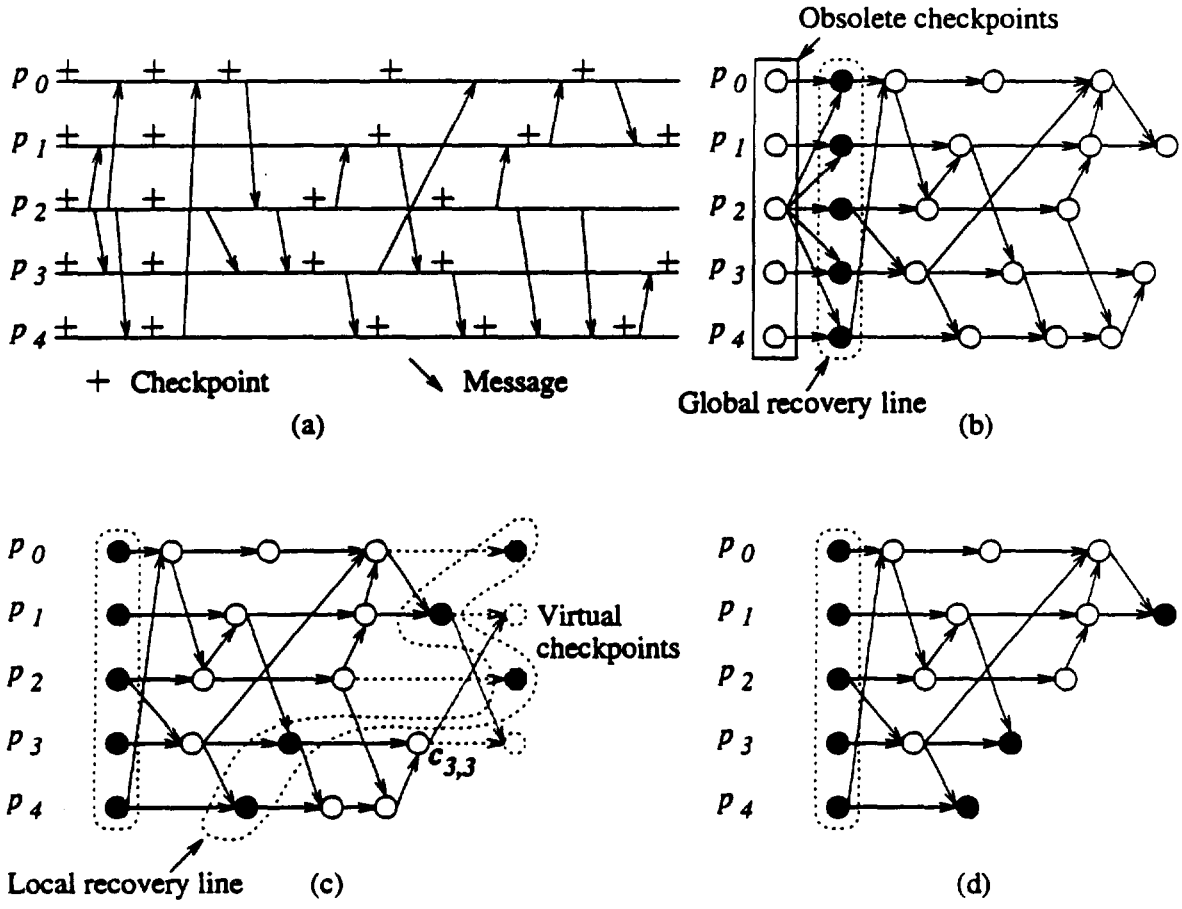


Figure 1.4: Checkpoint graphs and recovery lines. (a) Checkpoint and communication pattern; (b) checkpoint graph; (c) extended checkpoint graph; (d) checkpoint graph after recovery.

1.4 A Model of Consistent Global Checkpoints

1.4.1 Partially ordered sets, antichains and lattices

A *partial order* [45] on a set S is a relation "$<$" such that

(a) $\forall s \in S,\ s \not< s$. (Irreflexivity)

(b) If $s < t$, then $t \not< s$. (Antisymmetry)

```
/* CP represents a checkpoint */
/* Initially, all of 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 on the
    same process;
   mark all CPs strictly reachable from any CP in the root set;
}
the root set is the recovery line.
```

Figure 1.5: The rollback propagation algorithm.

(c) If $s < t$ and $t < u$, then $s < u$. (Transitivity)

The pair $(S, <)$ is called a *partially ordered set*, or *poset*. An element s is *minimal* if there does not exist any element w such that $w < s$. An element t of S is a *minimum element* if $t \leq w$ for all w in S. *Maximal* and *maximum* elements are similarly defined.

A subset H of S is a *chain* of $(S, <)$ if the elements of H can be enumerated as $h_1, h_2, \ldots h_n$ such that $h_1 < h_2 < \ldots < h_n$. A subset A of S is an *antichain* of $(S, <)$ if $a \not< b$ for all $a, b \in A$. An antichain M of $(S, <)$ with the largest size of any antichain is called a *maximum-sized antichain.*

Given two elements s and t of a poset $(S, <)$, we write $s \leq t$ if $s < t$ or $s = t$. Any element l such that $l \leq s$ and $l \leq t$ is called a *lower bound* of s and t. If there exists a lower bound l^* such that $l \leq l^*$ for all lower bounds l of s and t, then l^* is called the *greatest lower bound* of s and t. *Upper bound* and *least upper bound* are similarly defined. A *lattice* is a poset $(S, <)$ which possesses both a greatest lower bound (called the **meet**

and denoted by $s \wedge t$) and a least upper bound (called the **join** and denoted by $s \vee t$) for all $s, t \in S$.

Let $P = (S, <)$, and let $\mathcal{M}(P)$ denote the set of maximum-sized antichains of P. A partial order $\preceq$ on the maximum-sized antichains can be defined as follows [46]: for any $M_1, M_2 \in \mathcal{M}(P)$,

$$M_1 \preceq M_2 \text{ iff for every } a_1 \in M_1, \text{ there exists } a_2 \in M_2 \text{ such that } a_1 \leq a_2. \qquad (1.1)$$

It has been shown that [46, §13.1-13.2], for any poset P, $(\mathcal{M}(P), \preceq)$ forms a lattice and therefore possesses a unique maximum element called the *maximal maximum-sized antichain* and denoted by $M^*(P)$.

1.4.2 Consistent global checkpoints and recovery lines

The execution of each process in a message-passing system can be viewed as a sequence of *events*, corresponding to the state changes that take place in the process. The collection of event sequences for the participating processes forms the *execution history* of the system. The proper granularity at which to view "events" varies from application to application. For our purposes, the events of interest are the sending and receiving of messages, and the recording of local checkpoints by individual processes. An execution history restricted to these events will be called a *checkpoint and communication pattern.* We assume that the first event in each process is an initial local checkpoint event.

The global set of events appearing in a checkpoint and communication pattern cannot be placed naturally in a total order, as can the events of a single process. Instead, a partial

order on the events can be defined as follows. We say that event e_1 *directly happened before* event e_2 [19, 47], denoted by $e_1 <_d e_2$, if

1. e_1 and e_2 are events in the same process and e_1 occurs immediately before e_2; or

2. e_1 is the sending of a message m and e_2 is the receiving of m.

The transitive closure of the $<_d$ relation is the *happened before* relation $<$ [47].

Let $\mathcal{P}$ be a checkpoint and communication pattern. A *global checkpoint* of $\mathcal{P}$ is a set of N local checkpoints, one from each process. Based on the previous description of checkpoint consistency, two checkpoints are inconsistent if and only if they are ordered by the *happened before* relation. For example, $c_{j,y}$ and $c_{i,x+1}$ in Fig. 1.1(a) are inconsistent because $c_{j,y} < c_{i,x+1}$. A *consistent global checkpoint* of $\mathcal{P}$ is therefore a global checkpoint of which no two constituent checkpoints are ordered by the *happened before* relation. We will denote by $E_{\mathcal{P}}$ the set of events that appear in $\mathcal{P}$, and by $Q_{\mathcal{P}}$ the poset generated by the *happened before* relation on those events: $Q_{\mathcal{P}} = (E_{\mathcal{P}}, <)$. Let $R_{\mathcal{P}} = (C_{\mathcal{P}}, <)$ be the *induced subposet* [45] of $Q_{\mathcal{P}}$ obtained by restricting the $<$ relation to $C_{\mathcal{P}}$, the set of all checkpoints. In the remainder of this section we derive an important characterization of consistent global checkpoints related to the poset $R_{\mathcal{P}}$.

LEMMA 1 *The largest size of any antichain in $R_{\mathcal{P}}$ is N, and every antichain of size N includes a checkpoint from each process in $\mathcal{P}$.*

Proof. The initial checkpoints form an antichain of size N and hence the largest size of any antichain in $R_{\mathcal{P}}$ is at least N. Because any two checkpoints from the same process

must be ordered by the *happened before* relation, the largest size of any antichain is exactly N, and every antichain of size N must include a checkpoint from each process. □

THEOREM 1 *M is a consistent global checkpoint in $\mathcal{P}$ if and only if it is a maximum-sized antichain in $R_{\mathcal{P}}$.*

Proof. By definition, a consistent global checkpoint of $\mathcal{P}$ is clearly an antichain of size N in $Q_{\mathcal{P}}$ and therefore in $R_{\mathcal{P}}$ as well (since $R_{\mathcal{P}}$ is an induced subposet of $Q_{\mathcal{P}}$). By Lemma 1, it is a maximum-sized antichain in $R_{\mathcal{P}}$.

Conversely, if M is a maximum-sized antichain in $R_{\mathcal{P}}$, then by Lemma 1 it includes a local checkpoint from each process in $\mathcal{P}$ and these local checkpoints are pairwise unordered by $<$. Thus M is a consistent global checkpoint of $\mathcal{P}$. □

For a given antichain A of $R_{\mathcal{P}}$, we let $A[i]$ denote the element of A which is a checkpoint of process p_i. The following lemma shows that for the poset $R_{\mathcal{P}}$, Anderson's global $\preceq$ relation as defined in Eq. (1.1) reduces to local ordering of checkpoints within each process.

LEMMA 2 *For any $M_1, M_2 \in \mathcal{M}(R_{\mathcal{P}})$,*

$$M_1 \preceq M_2 \text{ iff } M_1[i] \leq M_2[i], \text{ for all } 0 \leq i \leq N-1. \tag{1.2}$$

Proof. Suppose $M_1 \preceq M_2$. For any given i let j be such that $M_1[i] \leq M_2[j]$ in $R_{\mathcal{P}}$. Since $M_1[i]$ and $M_2[i]$ are events in the same process, either $M_2[i] < M_1[i]$ or

$M_1[i] \leq M_2[i]$. In the first case, we would have $M_2[i] < M_1[i] \leq M_2[j]$, contradicting the fact that M_2 is an antichain in $R_{\mathcal{P}}$. Hence $M_1[i] \leq M_2[i]$, and this is true for all $0 \leq i \leq N-1$. Conversely, the assumption that $M_1[i] \leq M_2[i]$ for any i yields $M_1 \preceq M_2$ by definition of $\preceq$. □

From Lemma 2 we see that for any $M \in \mathcal{M}(R_{\mathcal{P}})$, $M[i] \leq M^*(R_{\mathcal{P}})[i]$ for all $0 \leq i \leq N-1$, and it follows that $M^*(R_{\mathcal{P}})$ corresponds to the consistent global checkpoint of $\mathcal{P}$ in which each constituent local checkpoint is as advanced as possible. The antichain $M^*(R_{\mathcal{P}})$ is therefore what we have referred to as the "recovery line" of $\mathcal{P}$ with the minimum total rollback distance [48].

Our development of the optimal garbage collection algorithm will be based on checkpoint graphs rather than the more abstract posets. Given a checkpoint graph G, we let $\mathcal{M}(G)$ denote the set of maximum-sized antichains of the poset $R^c_{\mathcal{P}}$ corresponding to the transitive closure of G. The maximal maximum-sized antichain $M^*(G)$ is similarly defined. We prove in Appendix A that although the two posets $R^c_{\mathcal{P}}$ and $R_{\mathcal{P}}$ are *not* the same due to some missing dependencies in $R^c_{\mathcal{P}}$ resulted from the direct dependency tracking mechanism, $R^c_{\mathcal{P}}$ possesses exactly the same set of maximum-sized antichains as does $R_{\mathcal{P}}$ and therefore suffices for our purpose.

2. OPTIMAL GARBAGE COLLECTION

In this chapter, we describe the approach of recovery line transformation and decomposition to achieving optimal garbage collection. The term "optimal" means we can identify all of the checkpoints and messages logs that are no longer useful for any future recovery, and all of the retained checkpoints and message logs must be useful for some possible future recovery. Section 2.1 derives the necessary and sufficient condition for identifying all garbage checkpoints, which then leads to an optimal checkpoint reclamation algorithm [49]. Section 2.2 derives the lowest upper bound on the number of nongarbage checkpoints. For protocols requiring message logging to record the in-transit messages, Section 2.3 derives the necessary and sufficient condition for identifying all garbage message logs and develops an optimal message log reclamation algorithm which can be combined with the optimal checkpoint reclamation algorithm to minimize the space overhead for uncoordinated checkpointing.

2.1 Optimal Checkpoint Reclamation

2.1.1 Motivation and problem formulation

Since a future program execution may contain arbitrary checkpoint dependencies and rollbacks, we first describe an execution model to make the problem tractable. An *operational session* [5] is the interval between the start of normal execution and the instance

of rollback initiation, as shown in Fig. 2.1. A *recovery session* immediately follows the previous operational session and ends at the resumption of normal execution. A program execution can be viewed as consisting of a number of alternating operational sessions and recovery sessions. In terms of the effect on the checkpoint graphs, new vertices are added as new checkpoints are taken during an operational session, and existing vertices can be deleted as some checkpoints are invalidated by the rollback during a recovery session.

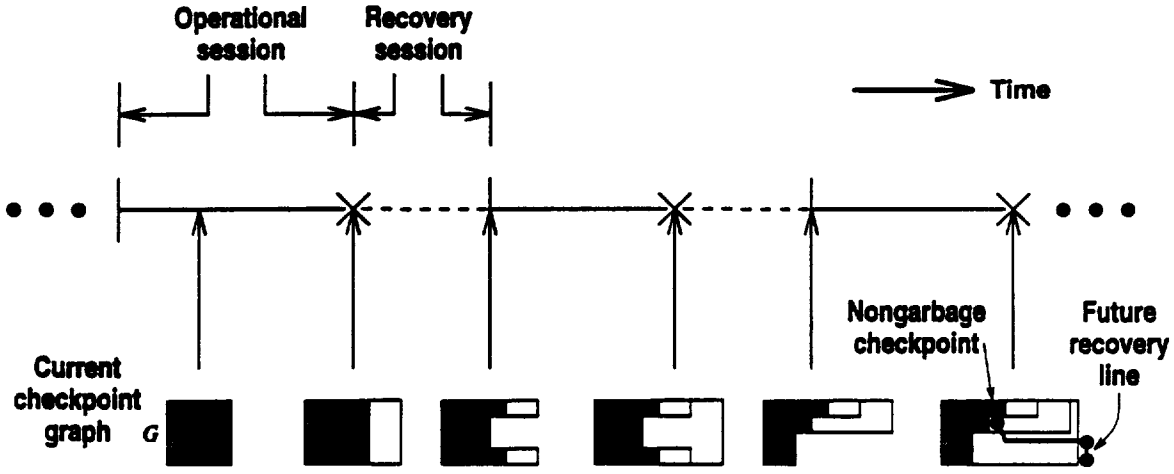


Figure 2.1: Operational sessions, recovery sessions and nongarbage checkpoints.

Since the purpose of maintaining checkpoints is for possible future recovery, a checkpoint is *garbage* if and only if it can not belong to any future recovery line. Being obsolete, i.e., before the global recovery line, is simply a sufficient condition for being garbage, but not a necessary condition. We first give an example of nonobsolete garbage checkpoints. Figure 2.2 is a typical example illustrating the domino effect. The global recovery line stays at the set of initial checkpoints and is unable to move forward. The edge from

$c_{0,2}$ to $c_{1,2}$ and the one from $c_{1,1}$ to $c_{0,2}$ imply that $c_{0,2}$ is inconsistent with any checkpoint of process p_1. Since a recovery line must contain one checkpoint from each process, $c_{0,2}$ can not belong to any future recovery line[1] and is therefore a garbage checkpoint. Checkpoints $c_{1,1}$ and $c_{0,1}$ are garbage by similar arguments.

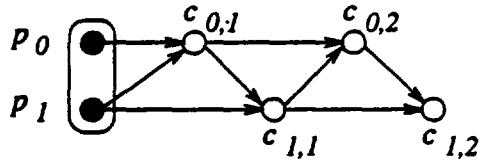


Figure 2.2: Example of nonobsolete garbage checkpoints.

Figure 2.2 in fact provides another sufficient condition for identifying garbage checkpoints; our optimal garbage collection aims at deriving the necessary and sufficient condition. The difficulty of the problem lies in the fact that future process execution may contain any number of operational sessions (with arbitrary checkpoint dependencies) and recovery sessions (with arbitrary subsets of processes being faulty). We outline our approach as follows. Instead of trying to find garbage checkpoints, we start with identifying nongarbage checkpoints. Given any possible future recovery line which contains some nongarbage checkpoints, for example, the recovery line shown in Fig 2.1, we perform *recovery line transformation* to transform it into another recovery line which also contains those nongarbage checkpoints. Although there are an infinite number of future recovery lines containing any nongarbage checkpoint, we prove that they can all be transformed into a set of 2^N "immediate future" recovery lines. (Recall that N is the number of

[1]It is not hard to see that $c_{0,2}$ being a garbage checkpoint will not be affected by the occurrence of any recovery session because every rollback either preserves the "triangular" condition in Fig. 2.2 for $c_{0,2}$ or simply invalidates $c_{0,2}$.

processes.) Our next step is *recovery line decomposition.* We identify a set of N recovery lines which forms the "basis" for those 2^N recovery lines and therefore contains all of the nongarbage checkpoints.

2.1.2 Recovery line transformation

Our approach to transforming an arbitrary future recovery line backwards in time is to first define two elementary transformations: *transformation within an operational session* and *transformation across a recovery session.* Any transformation can then be achieved through a combination of these two elementary transformations.

Transformation within an operational session

During normal process execution, the size of the checkpoint graph increases as new checkpoints are taken. Because checkpoint graphs represent program dependencies and are not arbitrary directed acyclic graphs, the following rules must be satisfied when adding new vertices. For every new vertex $c_{i,x}$ with $x \geq 1$,

Rule 1.a: $c_{i,x}$ must have an incoming edge from $c_{i,x-1}$;

Rule 1.b: $c_{i,x}$ can not have any outgoing edge to any existing vertices because it can not *happen before* a checkpoint that was taken earlier.

We note that because of the unpredictable message transmission delay during the dependency information collection process, the information associated with a checkpoint $c_{j,y}$ that *happened before* $c_{i,x}$ is not necessarily collected by the garbage collection initiator earlier than the information associated with $c_{i,x}$ is collected. However, such a situation

can be detected based on the dependency information. If a vertex $c_{i,x}$ is supposed to have an incoming edge from a nonexisting vertex $c_{j,y}$, then $c_{i,x}$ and all of its incoming edges will be temporarily excluded from the current checkpoint graph. By adding each new vertex under this constraint, none of the new vertices can have any edge pointing to any existing vertices and Rule 1.b is therefore enforced. We use $\mathcal{G}_s(G)$ to denote the set of all *potential supergraphs* obtainable by adjoining new vertices to a given checkpoint graph G without violating Rule 1.a and Rule 1.b.

Our transformation procedure generally involves changing part of the recovery line of a graph G_1 to obtain the recovery line of another graph G_2. The following lemma will be used throughout this chapter to ensure that the unchanged part, which forms an antichain in G_1, remains an antichain in G_2 after the transformation.

LEMMA 3 *Given a checkpoint graph $G = (V, E)$ and its potential supergraph $G' = (V', E') \in \mathcal{G}_s(G)$, for any $A \subseteq V$, A is an antichain in G if and only if A is an antichain in G'.*

Proof. If A is an antichain in G, then $u \not< v$ for any $u, v \in A$. Rule 1.b guarantees that $u \not< v$ remains true in G' because there can not exist any $w \in V' \setminus V$ such that $u < w < v$. Hence, A is an antichain in G'. Conversely, if A is not an antichain in G, there must exist $u, v \in A$ such that $u < v$ which clearly remains true in G', and so A is not an antichain in G'. □

One special potential supergraph of G, denoted by $\hat{G}$, will play a major role throughout this chapter. The graph $\hat{G}$ is constructed by adjoining a new vertex n_i at the end of

G for each p_i, with a single incoming edge from the last vertex l_i as shown in Fig 2.3. Let L denote the set of all *last-nodes* l_i and B denote the set of all *new-nodes* n_i. We will refer to the 2^N graphs $\hat{G} - W$, $W \subseteq B$, as the *immediate supergraphs* of G. The proof of the following property defines the recovery transformation within an operational session: given the recovery line of a potential supergraph G' of G, by replacing its constituent checkpoints *which are not contained in* G with their corresponding *new-nodes* of G, we obtain the recovery line of an immediate supergraph of G.

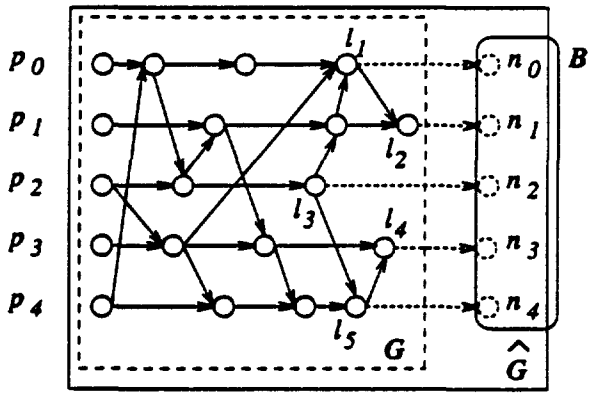


Figure 2.3: Construction of the potential supergraph $\hat{G}$.

PROPERTY 1 *For any checkpoint v in a checkpoint graph G, if v belongs to the recovery line of a potential supergraph G', then v must also belong to the recovery line of an immediate supergraph of G. That is, given $G = (V, E)$, $v \in V$ and $G' \in \mathcal{G}_s(G)$, if $v \in M^*(G')$, then $v \in M^*(\hat{G} - W)$ for some $W \subseteq B$.*

Proof. We partition $M^*(G')$ into $M_1 \cup M_2$ where

$$M_1 = M^*(G') \cap V$$

$$M_2 = M^*(G') \setminus V$$

as shown in Fig. 2.4. A corresponding partition of the *new-nodes* of G is given as $B = B_1 \cup B_2$ such that

$$B_1 = \{n_i : M^*(G')[i] \in M_1\}$$

$$B_2 = \{n_i : M^*(G')[i] \in M_2\}.$$

Our goal is to show that

$$M^*(\hat{G} - B_1) = M_1 \cup B_2.$$

Then, for any $v \in V$ and $v \in M^*(G')$, we must have $v \in M_1 \subseteq M^*(\hat{G} - W)$ where $W = B_1 \subseteq B$.

First, we show that $M_1 \cup B_2 \in \mathcal{M}(\hat{G} - B_1)$. Define the subset L_2 of *last-nodes* corresponding to M_2 as $L_2 = \{l_j : M^*(G')[j] \in M_2\}$. Because $M_1 \cup M_2$ forms an antichain in G', we must have $M^*(G')[i] \not\preceq l_j$ for any $M^*(G')[i] \in M_1$ and $l_j \in L_2$. Now consider $\hat{G} - B_1$. We have $M^*(G')[i] \not\preceq n_j$ for any $n_j \in B_2$ because each n_j has only a single incoming edge from l_j. Clearly, any *new-node* $n_j \not\preceq M^*(G')[i]$. Lemma 3 further guarantees that $M_1 (\subseteq V)$ remains an antichain in G and also in $\hat{G} - B_1$. Hence, we have $M_1 \cup B_2 \in \mathcal{M}(\hat{G} - B_1)$.

We next prove that $M_1 \cup B_2 = M^*(\hat{G} - B_1)$ by contradiction. Suppose $M_1 \cup B_2 \neq M^*(\hat{G} - B_1)$. There must exist $M_1' = M^*(\hat{G} - B_1) \setminus B_2$ such that $M_1' \subseteq V$, $M_1 \preceq M_1'$ and $M_1 \neq M_1'$ as shown in Fig. 2.4. Now consider G'. Recall that M_1 and M_2 form an antichain in G' and thus for any $u \in M_1'$ and $M^*(G')[j] \in M_2$, we must have $u \not\preceq M^*(G')[j]$. We also have $M^*(G')[j] \not\preceq u$ by Rule 1.b. Therefore, $M_1' \cup M_2$ forms an

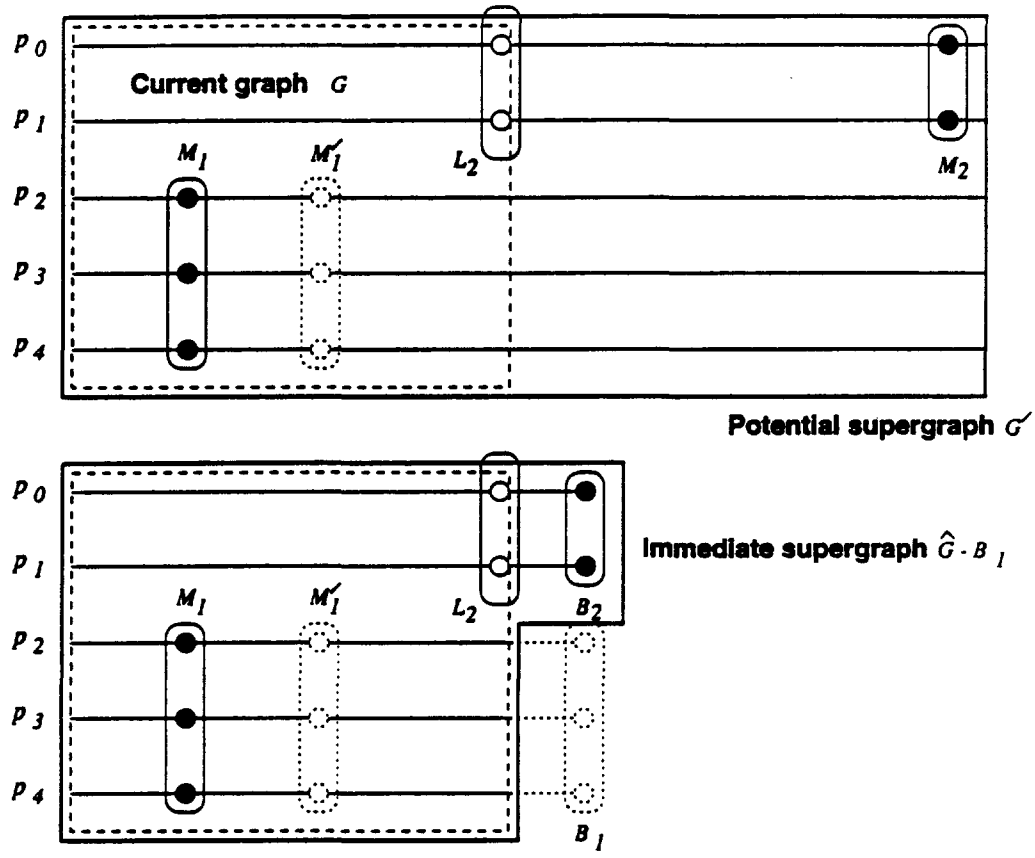


Figure 2.4: Recovery line transformation within an operational session.

antichain in G', contradicting the fact that $M_1 \cup M_2$ is the maximal maximum-sized antichain of G'. □

The transformation within an operational session can be viewed as "projecting" any potential supergraph along the direction opposite to the time axis. It shows that although the number of potential supergraphs of G is infinite, the recovery lines of these graphs can intersect G in only a finite number of ways, and each of the possible intersections must be part of the recovery line of an immediate supergraph of G.

Transformation across a recovery session

Existing vertices on a checkpoint graph, for example, $c_{3,3}$ in Fig. 1.4(c), can be deleted due to rollback recovery. Let G_E denote the extended checkpoint graph as defined in Section 1.3, $G = (V, E)$ denote the subgraph of G_E without the virtual checkpoints, and $G^- = (V^-, E^-)$ denote the checkpoint graph immediately after recovery. Figures 2.5(a)-(c) illustrate these checkpoint graphs. Let F denote the part of G deleted by the rollback; then we have $G^- = G - F$. By definition, $M^*(G_E)$ is the local recovery line. Let $M^*(G_E) = M_r \cup M_v$ as shown in Fig. 2.5(b) where $M_r = M^*(G_E) \cap V$ consists of real checkpoints and $M_v = M^*(G_E) \setminus M_r$ consists of **virtual** checkpoints. According to the rollback propagation algorithm, the following two rules must be satisfied when existing vertices are deleted during recovery.

Rule 2.a: There cannot exist any $u \in M_r$ and $w \in V^-$ such that $u < w$, i.e., none of the checkpoints in M_r can have any outgoing edge in G^-.

Rule 2.b: For any u in F, all of the checkpoints reachable by u must also be in F. Consequently, none of the checkpoints in F can have any outgoing edge to any checkpoints in G^-.

We also define

$$T_1 = \{n_i : M^*(G_E)[i] \in M_r\} \text{ and } T_2 = \{n_j : M^*(G_E)[j] \in M_v\}. \tag{2.1}$$

In other words, T_1 consists of the *new-node* n_i for each process p_i which contributes a real checkpoint to the local recovery line. Parallel to the definitions of l_i, n_i, B, $\hat{G}$, T_1 and T_2

for G, we define l_i^-, n_i^-, $\hat{G}^-$, B^-, T_1^- and T_2^- for G^-. That is, l_i^- denotes the *last-node* of p_i in G^-, n_i^- denotes the *new-node* of p_i in G^-, $\hat{G}^-$ is obtained by adding n_i^- to G^- for every p_i, B^- denotes the set of all *new-nodes* in $\hat{G}^-$, $T_1^- = \{n_i^- : M^*(G_E)[i] \in M_r\}$ and $T_2^- = \{n_j^- : M^*(G_E)[j] \in M_v\}$. It is not hard to see that $T_2^- = T_2$.

We first prove the following lemma which states the relationship between the maximum-sized antichains of G and those of its potential supergraphs.

LEMMA 4 *Given a checkpoint graph $G = (V, E)$ and its potential supergraph $G' = (V', E') \in \mathcal{G}_s(G)$, for any $M \subseteq V$,*

(a) *$M \in \mathcal{M}(G)$ if and only if $M \in \mathcal{M}(G')$;*

(b) *$M^*(G) \preceq M^*(G')$;*

(c) *if $M = M^*(G')$ then $M = M^*(G)$.*

Proof. Rule 1.a guarantees that the largest size of any antichain remains the same in all potential supergraphs. Hence, **(a)** follows immediately from Lemma 3. In particular, $M^*(G) \in \mathcal{M}(G')$ which leads to **(b)**. If $M \subseteq V$ and $M = M^*(G')$, then $M^*(G) \preceq M$ from **(b)** leads to $M = M^*(G)$. □

The proof of the following property defines the transformation across a recovery session: given the recovery line M of an immediate supergraph of G^-, for any i such that *$M[i]$ is a new-node and $M^*(G_E)[i]$ from the local recovery line is not a virtual checkpoint*, we replace $M[i]$ with $M^*(G_E)[i]$ to obtain the recovery line of an immediate supergraph of G.

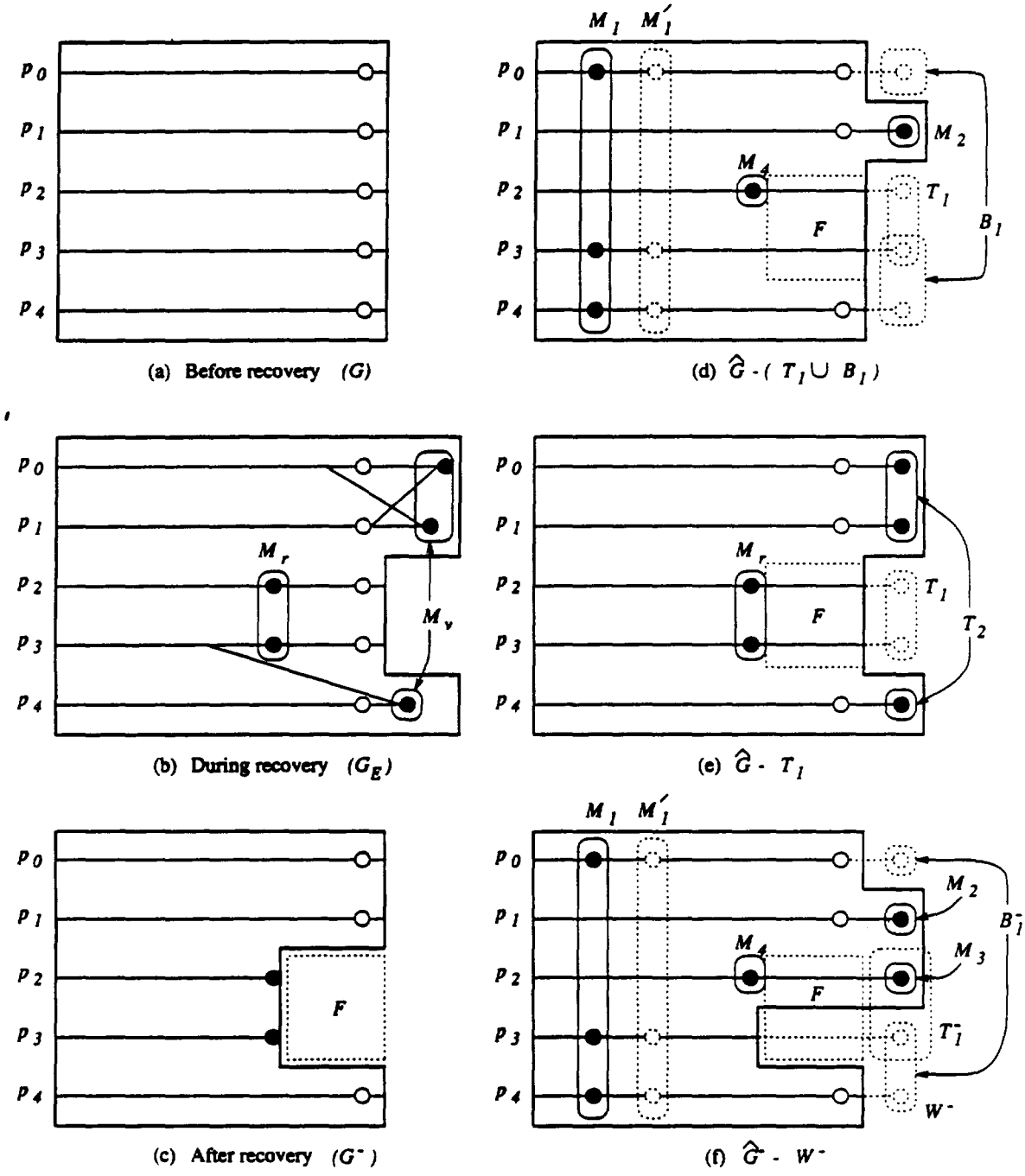


Figure 2.5: Recovery line transformation across a recovery session.

PROPERTY 2 *For any checkpoint v in G^-, if v belongs to the recovery line of an immediate supergraph of G^-, then v must also belong to the recovery line of an immediate supergraph of G. That is, given $G^- = (V^-, E^-)$ and $v \in V^-$, if $v \in M^*(\hat{G}^- - W^-)$ for some $W^- \subseteq B^-$, then $v \in M^*(\hat{G} - W)$ for some $W \subseteq B$.*

Proof. Let $G^-_W = \hat{G}^- - W^-$. We partition the recovery line $M^*(G^-_W)$ into $M_1 \cup M_2 \cup M_3$ where

$$\begin{aligned} M_1 &= M^*(G^-_W) \cap V^- \\ M_2 &= \{n_i^- \in M^*(G^-_W) : M^*(G_E)[i] \in M_v\} \\ M_3 &= \{n_i^- \in M^*(G^-_W) : M^*(G_E)[i] \in M_r\} \end{aligned} \tag{2.2}$$

as shown in Fig. 2.5(f). The two sets of *new-nodes* B and B^- are partitioned[2] as follows.

$$B = B_1 \cup B_2 \quad \textit{where} \quad B_1 = \{n_i : M^*(G^-_W)[i] \in M_1\} \tag{2.3}$$
$$B_2 = \{n_i : M^*(G^-_W)[i] \notin M_1\}$$

$$B^- = B_1^- \cup B_2^- \quad \textit{where} \quad B_1^- = \{n_i^- : M^*(G^-_W)[i] \in M_1\} \tag{2.4}$$
$$B_2^- = \{n_i^- : M^*(G^-_W)[i] \notin M_1\}.$$

Our goal is to show that

$$M_1 \cup M_2 \cup M_4 = M^*(\hat{G} - (T_1 \cup B_1)) \tag{2.5}$$

where

$$M_4 = \{M^*(G_E)[i] : n_i^- \in M_3\}.$$

[2] $T_1 \cup T_2$ (Eq. (2.1)) is another partition of B corresponding to $M^*(G_E) = M_r \cup M_v$.

Then, for any $v \in V^-$ and $v \in M^*(G_W^-)$, we have $v \in M_1 \subseteq M^*(\hat{G} - W)$ where $W = T_1 \cup B_1 \subseteq B$.

First, it is not hard to see that $W^- \subseteq B_1^-$ and so $M^*(\hat{G}^- - B_1^-) = M^*(G_W^-)$ from Lemma 4(c) and the definitions of B_1^- and M_1. We now prove Eq. (2.5) by the following steps: **(a)** $M_1 \cup M_2 \cup M_4 \in \mathcal{M}(\hat{G}^- - (T_1^- \cup B_1^-))$; **(b)** $M_1 \cup M_2 \cup M_4 \in \mathcal{M}(\hat{G} - (T_1 \cup B_1))$; **(c)** $M_1 \cup M_2 \cup M_4 = M^*(\hat{G} - (T_1 \cup B_1))$.

(a) That $M_1 \cup M_2 \cup M_3$ forms an antichain in $\hat{G}^- - B_1^-$ implies, for any $u \in M_4$ and $w \in M_1 \cup M_2$, that $w \not< u$. This clearly remains true in $\hat{G}^- - (T_1^- \cup B_1^-)$. Since Rule 2.a guarantees $u \not< w$, we have $M_1 \cup M_2 \cup M_4 \in \mathcal{M}(\hat{G}^- - (T_1^- \cup B_1^-))$.

(b) By adding all of the vertices in F to $\hat{G}^- - (T_1^- \cup B_1^-)$, we obtain the graph $\hat{G} - (T_1 \cup B_1)$ as shown in Fig. 2.5(d). Rule 2.b guarantees that the above process will not make any unordered pair in $\hat{G}^- - (T_1^- \cup B_1^-)$ become ordered. Therefore, $M_1 \cup M_2 \cup M_4$ remains a maximum-sized antichain in $\hat{G} - (T_1 \cup B_1)$.

(c) Suppose that $M_1 \cup M_2 \cup M_4 \neq M^*(\hat{G} - (T_1 \cup B_1))$. Then, we must have

$$M_1 \cup M_2 \cup M_4 \preceq M^*(\hat{G} - (T_1 \cup B_1)). \tag{2.6}$$

By applying the transformation as defined in the proof of Property 1 to graphs G and G_E (Fig. 2.5(b)), we have

$$M^*(\hat{G} - T_1) = M_r \cup T_2, \tag{2.7}$$

as shown in Fig. 2.5(e). Since $\hat{G} - T_1$ is a potential supergraph of $\hat{G} - (T_1 \cup B_1)$, we have, by Lemma 4(b),

$$M^*(\hat{G} - (T_1 \cup B_1)) \preceq M^*(\hat{G} - T_1). \tag{2.8}$$

Equations (2.6), (2.7) and (2.8) and the fact that $M_2 \subseteq T_2$ and $M_4 \subseteq M_r$ imply $M_2 \cup M_4 \subseteq M^*(\hat{G} - (T_1 \cup B_1))$, and there exists M_1' such that $M_1' = M^*(\hat{G} - (T_1 \cup B_1)) \setminus (M_2 \cup M_4)$, $M_1 \preceq M_1'$ and $M_1 \neq M_1'$ (as shown in Fig. 2.5(d)). Equations (2.7) and (2.8) further guarantee that M_1' does not intersect F and so must exist in $\hat{G}^- - (T_1^- \cup B_1^-)$ and hence $\hat{G}^- - B_1^-$. Following the same argument as in the last part of the proof of Property 1, we can show that the existence of M_1' leads to a contradiction to the fact that $M_1 \cup M_2 \cup M_3 = M^*(\hat{G}^- - B_1^-)$. □

Complete transformation

We now apply Properties 1 and 2 to transforming an arbitrary future recovery line containing some nongarbage checkpoints. By repeatedly applying Property 1 within every operational session and Property 2 across every recovery session, we demonstrate that every such future recovery line of G can be transformed into the recovery line of an immediate supergraph of G which preserves all of those nongarbage checkpoints.

PROPERTY 3 *If a checkpoint in G belongs to a future recovery line, then it must also belong to the recovery line of an immediate supergraph of G. That is, given $G = (V, E)$ and $v \in V$, if $v \in M^*(G')$ for a future checkpoint graph G', then $v \in M^*(\hat{G} - W)$ for some $W \subseteq B$.*

Proof. Without loss of generality, we may assume G is in the qth operational session and G' belongs to the rth session where $r \geq q$. Let G_i denote the checkpoint graph at the *end* of the ith operational session, G_i^- denote the checkpoint graph at the *beginning*

of the same session, and W_i denote a subset of *new-nodes* of G_i. Clearly, v must belong to every such intermediate graph. By applying Property 1 to the graph pairs (G', G_r^-), $(G_j - W_j, G_j^-)$ where $q+1 \leq j \leq r-1$ and $(G_q - W_q, G)$, and applying Property 2 to the graph pairs (G_j^-, G_{j-1}) where $q+1 \leq j \leq r$, we can show that v must always remain on the recovery line of an immediate supergraph of one of the intermediate graphs throughout the transformation procedure. Eventually, we have $v \in M^*(\hat{G} - W)$ for some $W \subseteq B$. □

Figure 2.6 gives an example demonstrating the recovery line transformation. Figure 2.6(a) is the current checkpoint graph G considered for garbage collection. Suppose that Fig. 2.6(b) is the extended checkpoint graph when p_3 initiates a rollback, then Figure 2.6(c) is the checkpoint graph immediately after the recovery. Fig. 2.6(d) shows another possible extended checkpoint graph when p_0 initiates a second rollback. Since checkpoints A and B are needed for recovery in this case, they should be considered nongarbage checkpoints of G. We first apply Property 1 to the graph pairs (G_d, G_c) and transform the recovery line of G_d into the recovery line of G_g (an immediate supergraph of G_c) by replacing X, Y and Z with their corresponding *new-nodes* of G_c, namely, P, Q and R, respectively. Then we apply Property 2 to the pair (G_c, G_b). Since p_3 and p_4 contribute real checkpoints C and D, respectively, to the local recovery line in Fig. 2.6(b), the recovery line of G_g is transformed into the recovery line of G_f (an immediate supergraph of G_b) by replacing Q and R with C and D. Finally, by applying Property 1 to the

pair (G_f, G), we obtain the recovery line of G_e (an immediate supergraph of G) which still contains the nongarbage checkpoints A and B.

2.1.3 Recovery line decomposition

Property 3 states that the recovery lines of the 2^N immediate supergraphs of G contain all nongarbage checkpoints. We next show that there exists a set of N recovery lines which forms a "basis" for the 2^N recovery lines: each of the 2^N recovery lines is the set of minimal elements of the union of a subset of the N basis recovery lines. Therefore, it suffices to find these N recovery lines to identify all nongarbage checkpoints.

Let $X \wedge Y$ denote the **meet** (greatest lower bound) of X and Y in a lattice and $\min(S)$ denote the set of minimal elements in S. We first show that the greatest lower bound of any k maximum-sized antichains can be obtained as the set of minimal elements in their union.

LEMMA 5 *Given a poset P, $M \in \mathcal{M}(P)$ and $M \preceq M_i \in \mathcal{M}(P)$ for $0 \leq i \leq k-1$ for any finite k, define $\bigwedge_{0 \leq i \leq k-1} M_i = (...((M_0 \wedge M_1) \wedge M_2) \ ... \ ) \wedge M_{k-1}$, then*

$$(a) \ \ M \preceq \bigwedge_{0 \leq i \leq k-1} M_i \in \mathcal{M}(P) \ \ \textit{and} \ \ (b) \ \ \bigwedge_{0 \leq i \leq k-1} M_i = \min\Big(\bigcup_{0 \leq i \leq k-1} M_i \Big).$$

Proof. Both parts will be proved by induction on k and based on the following theorem from Anderson's book [46]: for any poset Q and $M_1, M_2 \in \mathcal{M}(Q)$, the **meet** (greatest lower bound) of M_1 and M_2 can be expressed as

$$M_1 \wedge M_2 = \min(M_1 \cup M_2). \tag{2.9}$$

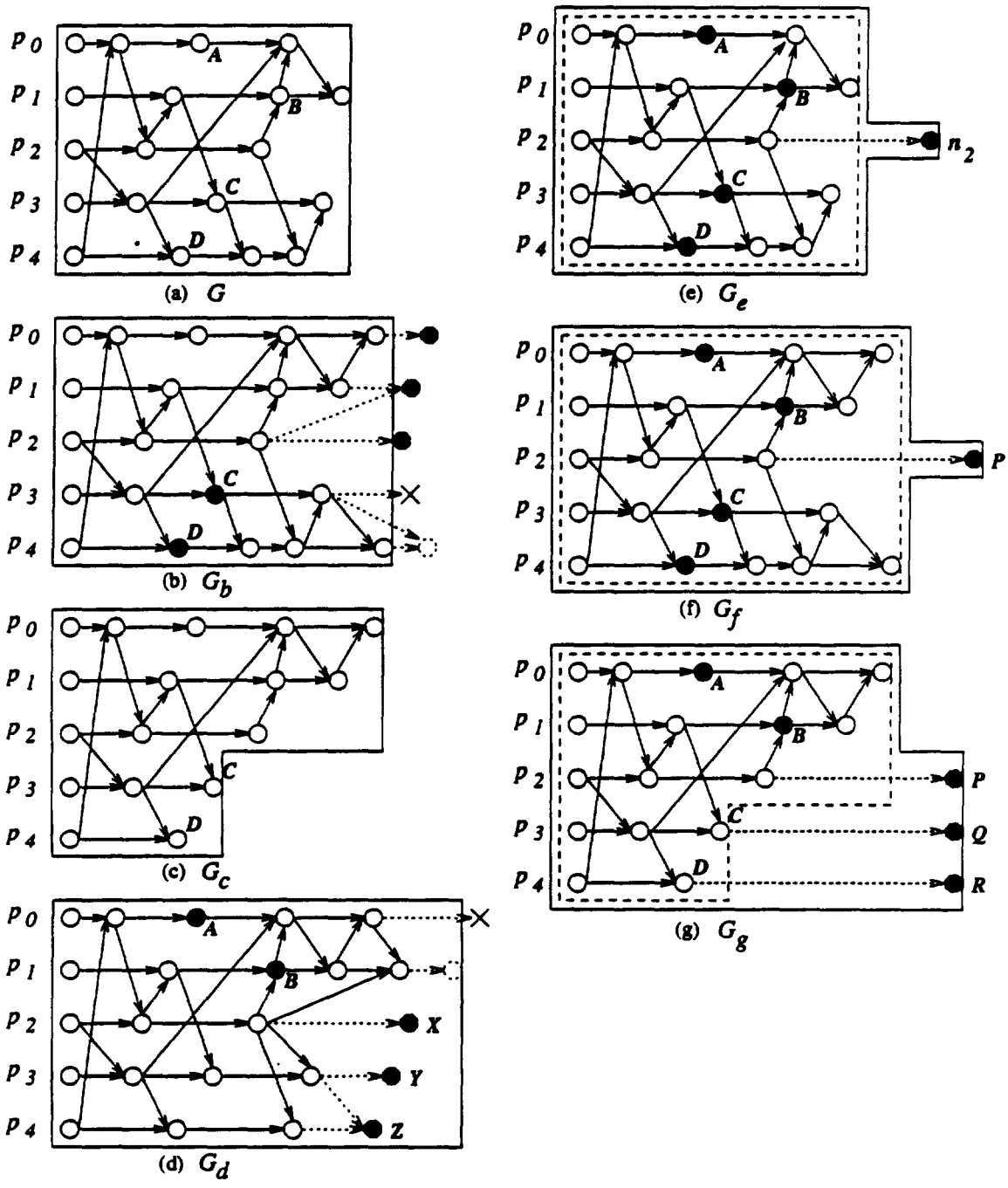


Figure 2.6: Example recovery line transformation.

(a) $\mathcal{M}(P)$ is a lattice and therefore $M_0 \wedge M_1 \in \mathcal{M}(P)$. Also, $M \preceq M_0 \wedge M_1$ because $M \preceq M_0$, $M \preceq M_1$ and $M_0 \wedge M_1$ is the greatest lower bound of M_0 and M_1. We have shown the case $k = 2$ is true. Assume that it is true for $k = n - 1$, i.e.,

$$M \preceq \bigwedge_{0 \le i \le n-2} M_i \in \mathcal{M}(P). \tag{2.10}$$

Again, the lattice property of $\mathcal{M}(P)$ ensures that

$$\bigwedge_{0 \le i \le n-1} M_i = (\bigwedge_{0 \le i \le n-2} M_i) \wedge M_{n-1} \in \mathcal{M}(P).$$

Equation (2.10) and $M \preceq M_{n-1}$ imply that

$$M \preceq \bigwedge_{0 \le i \le n-1} M_i.$$

Therefore, it is also true for $k = n$ and hence we have (a).

(b) The case $k = 2$ follows directly from Eq. (2.9). Assume that it is true for $k = n - 1$, i.e.,

$$\bigwedge_{0 \le i \le n-2} M_i = \min(\bigcup_{0 \le i \le n-2} M_i). \tag{2.11}$$

Applying part (a), Eqs. (2.9) and (2.11) , we have

$$\bigwedge_{0 \le i \le n-1} M_i = (\bigwedge_{0 \le i \le n-2} M_i) \wedge M_{n-1} = \min(\min(\bigcup_{0 \le i \le n-2} M_i) \cup M_{n-1}).$$

Lemma 6 (to be proved next) further gives that

$$\min(\min(\bigcup_{0 \le i \le n-2} M_i) \cup M_{n-1}) = \min(\bigcup_{0 \le i \le n-2} M_i \cup M_{n-1}) = \min(\bigcup_{0 \le i \le n-1} M_i).$$

Therefore, by induction, part (b) is true. □

LEMMA 6 *Given a poset $P = (S, <)$ and $X, Y \subseteq S$,* $\min(X \cup Y) = \min(\min(X) \cup Y)$.

Proof. First, we prove $\min(X \cup Y) \subseteq \min(\min(X) \cup Y)$. For every $z \notin \min(\min(X) \cup Y)$, there exists a z' in $\min(X) \cup Y$ such that $z' < z$. Since both z and z' are in $X \cup Y$, it follows that $z \notin \min(X \cup Y)$.

Conversely, we prove $\min(\min(X) \cup Y) \subseteq \min(X \cup Y)$. For every $z \notin \min(X \cup Y)$, there exists a z' in $X \cup Y$ such that $z' < z$. If $z' \in \min(X) \cup Y$, then we immediately obtain $z \notin \min(\min(X) \cup Y)$. Otherwise, if $z' \in X \setminus \min(X)$, then there exists a $z'' \in \min(X)$ such that $z'' < z'$, hence $z'' < z$, and again we have $z \notin \min(\min(X) \cup Y)$. □

PROPERTY 4 *For every $W \subseteq B$ and $W \neq \emptyset$,*

$$M^*(\hat{G} - W) = \min\Big(\bigcup_{n_i \in W} M^*(\hat{G} - n_i)\Big). \tag{2.12}$$

Proof. Without loss of generality, let $W = \{n_0, n_1, ..., n_{k-1}\}$ where $1 \leq k \leq N$. Since $\hat{G} - n_j \in \mathcal{G}_s(\hat{G} - W)$, $M^*(\hat{G} - W) \preceq M^*(\hat{G} - n_j)$ for all $0 \leq j \leq k-1$ by Lemma 4(b).

Now consider the graph $\hat{G}$. From Lemma 4(a), we have $M^*(\hat{G} - W) \in \mathcal{M}(\hat{G})$ and $M^*(\hat{G} - n_j) \in \mathcal{M}(\hat{G})$ for all $0 \leq j \leq k-1$. Let $M_n^* = \min(\bigcup_{0 \leq j \leq k-1} M^*(\hat{G} - n_j))$. From Lemma 5, we have

$$M^*(\hat{G} - W) \preceq \bigwedge_{0 \leq j \leq k-1} M^*(\hat{G} - n_j) \; = M_n^* \in \mathcal{M}(\hat{G}). \tag{2.13}$$

Since $M^*(\hat{G} - n_j)[j] < n_j$ and thus $n_j \notin M_n^*$ for all $0 \leq j \leq k-1$, every $x \in M_n^*$ must be contained in $\hat{G} - W$. From Lemma 4(a), we have $M_n^* \in \mathcal{M}(\hat{G} - W)$ and hence

$$M_n^* \preceq M^*(\hat{G} - W). \tag{2.14}$$

Combining Eqs. (2.13) and (2.14), we have proved that

$$M^*(\hat{G} - W) = M_n^* = \min(\bigcup_{n_i \in W} M^*(\hat{G} - n_i)).$$

□

In particular, the global recovery line $M^*(G)$ can be obtained by letting $W = B$, that is,

$$M^*(G) = \min(\bigcup_{0 \le i \le N-1} M^*(\hat{G} - n_i)).$$

As an example, we demonstrate the decomposition of $M^*(G_e)$ in Fig. 2.6(e) where $G_e = \hat{G} - \{n_0, n_1, n_3, n_4\}$. From Property 4 and referring to Fig. 2.7, we have

$$\begin{aligned} M^*(G_e) &= \min(M^*(\hat{G} - n_0) \cup M^*(\hat{G} - n_1) \cup M^*(\hat{G} - n_3) \cup M^*(\hat{G} - n_4)) \\ &= \min(\{A, B, n_2, n_3, n_4, n_0, I, n_1, J, C, D\}) = \{A, B, n_2, C, D\} \end{aligned}$$

which is exactly the recovery line shown in Fig. 2.6(e).

2.1.4 Predictive checkpoint space reclamation algorithm

We are now prepared to derive a necessary and sufficient condition for identifying all nongarbage checkpoints.

THEOREM 2 *A checkpoint v in a checkpoint graph G is nongarbage if and only if $v \in M^*(\hat{G} - n_i)$ for some $0 \le i \le N - 1$.*

Proof. If $v \in M^*(\hat{G} - n_i)$ for some $0 \le i \le N - 1$, then v is nongarbage because $\hat{G} - n_i$ is a possible future checkpoint graph. Conversely, if v is nongarbage, we have by definition

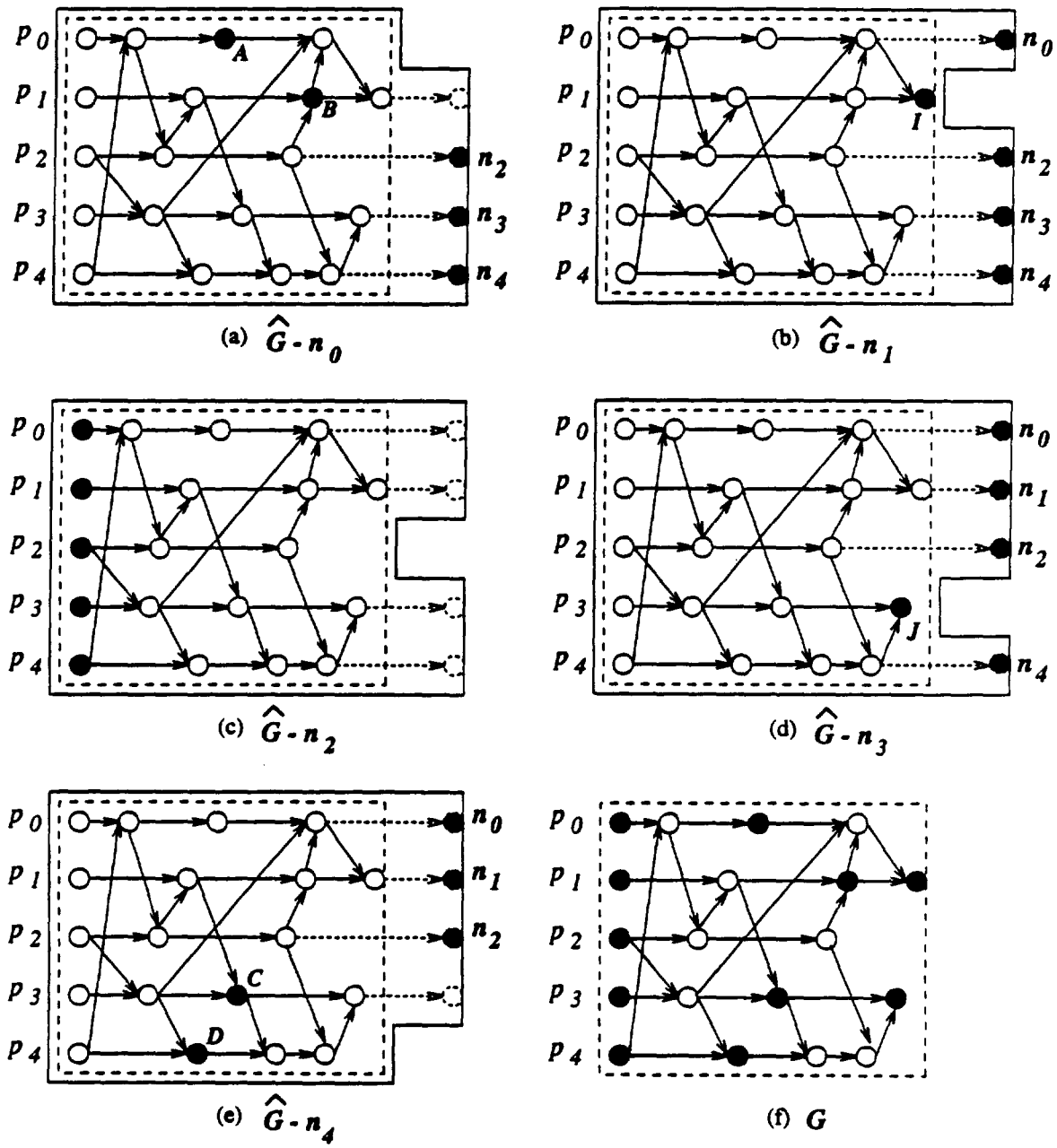


Figure 2.7: Example of the PCSR algorithm. Shaded checkpoints in (a)-(e) belong to the recovery lines and the nonshaded checkpoints in (f) are garbage.

$v \in M^*(G')$ for some future checkpoint graph G'. From Property 3, $v \in M^*(\hat{G} - W)$ for some $W \subseteq B$; from Property 4,

$$v \in \min(\bigcup_{n_i \in W} M^*(\hat{G} - n_i)) \subseteq \bigcup_{n_i \in W} M^*(\hat{G} - n_i) \subseteq \bigcup_{0 \le i \le N-1} M^*(\hat{G} - n_i).$$

Therefore, $v \in M^*(\hat{G} - n_i)$ for some $0 \le i \le N-1$. □

Based on Theorem 2 we now present the *Predictive Checkpoint Space Reclamation (PCSR)* algorithm for finding the N recovery lines in Fig. 2.8. Since the rollback propagation algorithm in Fig. 1.5 is of time complexity $\mathbf{O}(|E|)$ where $|E|$ is the total number of edges in the checkpoint graph (as every edge visited can be deleted), the PCSR algorithm is of time complexity $\mathbf{O}(N|E|)$.

/ $N_g(G)$ denotes the set of nongarbage checkpoints of G */*
/ N is the number of processes */*
/ $\hat{G}$ and n_i are as defined in Fig. 2.3 */*
for each $0 \le i \le N-1$ {
  apply the rollback propagation algorithm in Fig. 1.5 to the checkpoint graph $\hat{G} - n_i$ *to find the recovery line;*
  all checkpoints in the recovery line except for the new-nodes are included in the set $N_g(G)$*;*
}
all of the checkpoints not in $N_g(G)$ *can be garbage-collected.*

Figure 2.8: The Predictive Checkpoint Space Reclamation algorithm.

An example illustrating the execution of the PCSR algorithm on the checkpoint graph G in Fig. 2.3 is shown in Fig. 2.7. All of the checkpoints in G are nonobsolete and must be retained according to the traditional algorithm. Our PCSR algorithm, however, determines that all of the nonshaded checkpoints in Fig. 2.7(f) can be discarded.

2.1.5 Experimental results

Four parallel programs are used to illustrate the checkpoint space reclamation capabilities and benefits of the PCSR algorithm. Two of them are CAD programs written for Intel iPSC/2 hypercube: Cell Placement and Channel Router; the other two are Knight Tour and N-Queen written in the *Chare Kernel* language, which has been developed as a medium-grained machine-independent parallel language [50]. We use the Encore Multimax 510 multiprocessor version of the Chare Kernel. Communication traces are collected for these four programs, and trace-driven simulation is performed to obtain the results. The checkpoint interval for each program is arbitrarily chosen to be approximately ten percent of the total execution time, as shown in Table 2.1.

Table 2.1: Execution and checkpoint parameters of the programs.

Benchmark programs	Cell Placement	Channel Router	Knight Tour	N-Queen
Number of processors	8	8	6	6
Machine	Intel iPSC/2 hypercube	Intel iPSC/2 hypercube	Encore Multimax	Encore Multimax
Execution time (sec)	322.7	469.3	273.2	1625.1
Checkpoint interval (sec)	35	40	30	150

Figures 2.9-2.12 compare our PCSR algorithm with the traditional algorithm for typical executions of the four programs. Each curve shows the number of checkpoints which would be retained if the algorithm is invoked after a certain number of checkpoints

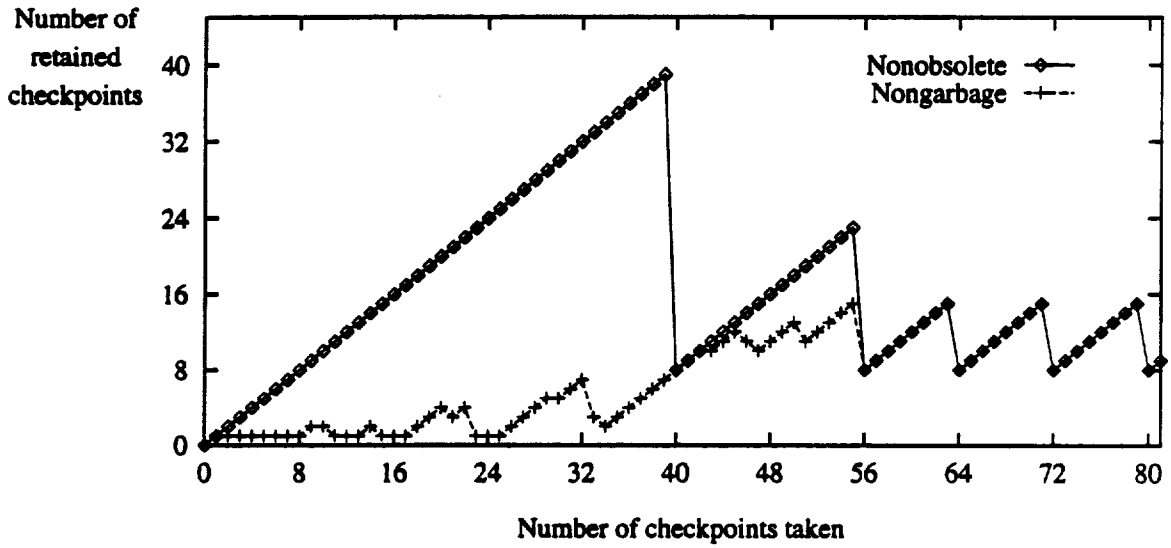


Figure 2.9: Nonobsolete and nongarbage checkpoints for Cell Placement.

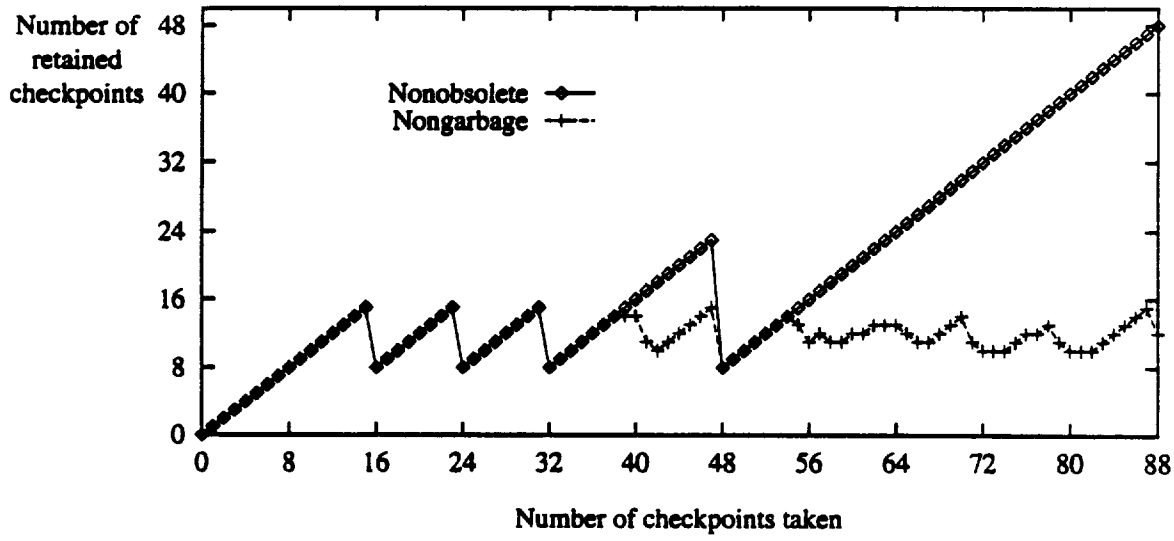


Figure 2.10: Nonobsolete and nongarbage checkpoints for Channel Router.

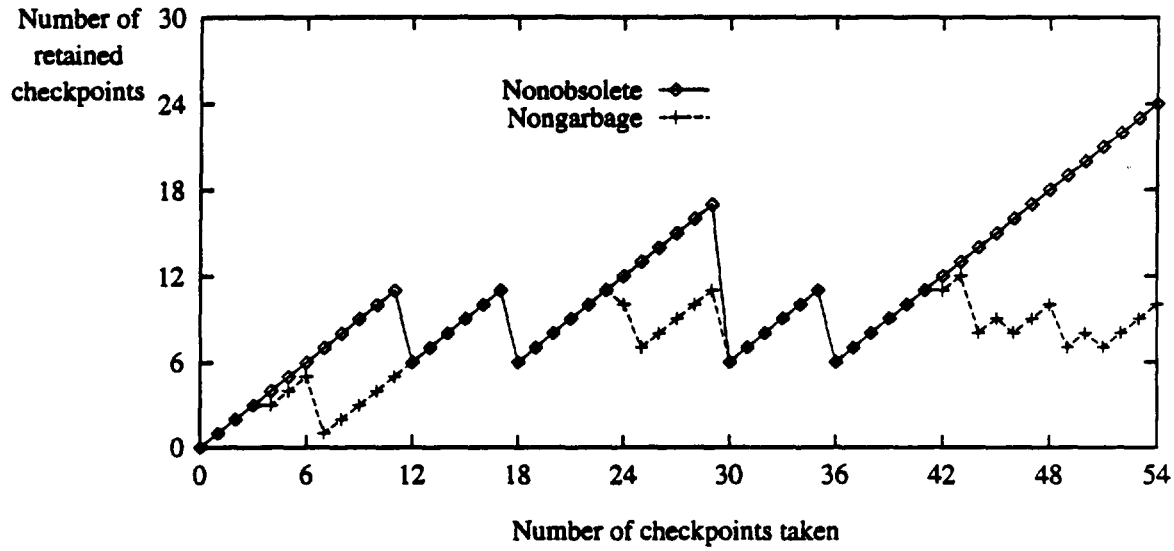


Figure 2.11: Nonobsolete and nongarbage checkpoints for Knight Tour.

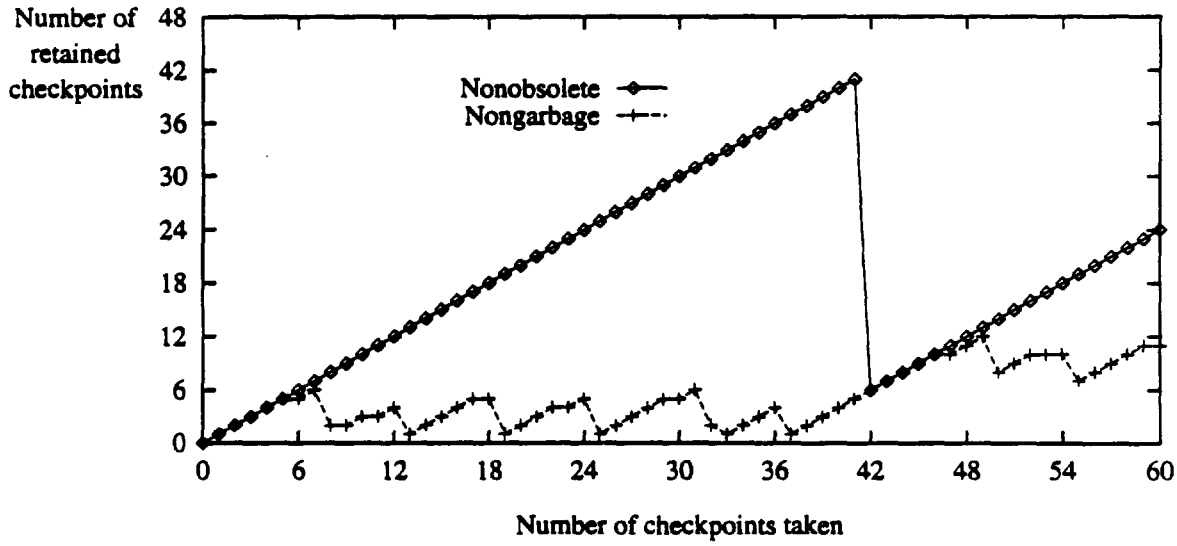


Figure 2.12: Nonobsolete and nongarbage checkpoints for N-Queen.

have been taken. The domino effect is illustrated by the linear increase in the number of nonobsolete checkpoints as the total number of checkpoints increases. The largest difference between the number of nonobsolete checkpoints and the number of nongarbage checkpoints for each program is 39 versus 7 for Cell Placement, 48 versus 12 for Channel Router, 24 versus 10 for Knight Tour and 41 versus 5 for N-Queen. .

2.2 Upper Bound on the Number of Nongarbage Checkpoints

As mentioned in Chapter 1, the traditional approach to garbage collection by discarding only obsolete checkpoints has lead to the common perception that the space overhead for uncoordinated checkpointing is potentially unbounded. Theorem 2 not only identifies the minimum set of nongarbage checkpoints but also places an upper bound N^2 on the number of nongarbage checkpoints because each $M^*(\hat{G} - n_i)$, $0 \leq i \leq N-1$, consists of N checkpoints. The following property identifies the inherent relations among $M^*(\hat{G} - n_i)$'s, and is the key to further improving the N^2 upper bound to the lowest upper bound $N(N+1)/2$.

PROPERTY 5 *For any* $0 \leq i, j \leq N-1$ *and* $i \neq j$, *if* $M^*(\hat{G} - n_i)[j] \neq n_j$ *and* $M^*(\hat{G} - n_j)[i] \neq n_i$, *then* $M^*(\hat{G} - n_i) = M^*(\hat{G} - n_j)$.

Proof. From Lemma 4(a), $M^*(\hat{G}-n_i)[j] \neq n_j$ implies $M^*(\hat{G}-n_i) \in \mathcal{M}(\hat{G}-n_i-n_j)$. Again from Lemma 4(a), $M^*(\hat{G} - n_i) \in \mathcal{M}(\hat{G} - n_j)$. We then have $M^*(\hat{G} - n_i) \preceq M^*(\hat{G}-n_j)$. Similarly, $M^*(\hat{G}-n_j)[i] \neq n_i$ leads to $M^*(\hat{G}-n_j) \preceq M^*(\hat{G}-n_i)$. Therefore, $M^*(\hat{G} - n_i) = M^*(\hat{G} - n_j)$. □

We note that the efficiency of the PCSR algorithm can be further improved by applying Property 5. Suppose that, inside the loop in Fig. 2.8, we have found the recovery line $M^*(\hat{G} - n_i)$ for all $0 \leq i \leq K$. Define the index set $\Gamma[j]$ for any $j > K$ as

$$\Gamma[j] = \{i : M^*(\hat{G} - n_i)[j] \neq n_j, 0 \leq i \leq K\}.$$

Then, for each later loop index j, the rollback propagation algorithm can be aborted when any checkpoint of process p_i, $i \in \Gamma[j]$, is marked. Because that would mean $M^*(\hat{G} - n_j)[i] \neq n_i$ and $M^*(\hat{G} - n_j)$ is exactly the same as $M^*(\hat{G} - n_i)$ by Property 5.

We are now prepared to prove the second major result.

THEOREM 3 *Let $N_g(G)$ denote the set of nongarbage checkpoints of G and N be the number of processes. Then,*

$$|N_g(G)| \leq \frac{N(N+1)}{2}.$$

Proof. By Theorem 2, we have to consider only the N^2 vertices $M^*(\hat{G} - n_i)[j]$, $0 \leq i, j \leq N-1$. First, $M^*(\hat{G} - n_i)[i]$ for all $0 \leq i \leq N-1$ must be in G and must contribute N vertices to $N_g(G)$. For the remaining $N^2 - N$ vertices with $i \neq j$, we consider the pair $M^*(\hat{G} - n_i)[j]$ and $M^*(\hat{G} - n_j)[i]$ one at a time and there are $(N^2 - N)/2$ such pairs. We distinguish three cases:

Case 1: $M^*(\hat{G} - n_i)[j] = n_j$ and $M^*(\hat{G} - n_j)[i] = n_i$. Both *new-nodes* do not belong to $N_g(G)$.

Case 2: $M^*(\hat{G} - n_i)[j] = n_j$ and $M^*(\hat{G} - n_j)[i] \neq n_i$, or $M^*(\hat{G} - n_i)[j] \neq n_j$ and $M^*(\hat{G} - n_j)[i] = n_i$. This pair will possibly add one new checkpoint to $N_g(G)$.

Case 3: $M^*(\hat{G}-n_i)[j] \neq n_j$ and $M^*(\hat{G}-n_j)[i] \neq n_i$. It follows that $M^*(\hat{G}-n_i) = M^*(\hat{G}-n_j)$ from Property 5, and thus $M^*(\hat{G}-n_i)[j] = M^*(\hat{G}-n_j)[j]$ and $M^*(\hat{G}-n_j)[i] = M^*(\hat{G}-n_i)[i]$. Since $M^*(\hat{G}-n_j)[j]$ and $M^*(\hat{G}-n_i)[i]$ are already in $N_g(G)$, this case does not increase the size of $N_g(G)$.

Therefore, each of the $(N^2-N)/2$ pairs can contribute at most one new checkpoint to $N_g(G)$ and hence

$$|N_g(G)| \leq N + \frac{N^2-N}{2} \times 1 = \frac{N(N+1)}{2}.$$

□

We next show that $N(N+1)/2$ is in fact the lowest upper bound because for any N we can construct a checkpoint graph G_N^* as shown in Fig. 2.13 to achieve this upper bound. Figure 2.13 shows the nongarbage checkpoints contributed by each of the N recovery lines in the PCSR algorithm. All of the $N(N+1)/2$ checkpoints are identified as nongarbage checkpoints.

As a final note, the greatest lower bound of N is achieved when none of the $(N^2-N)/2$ pairs contributes any nongarbage checkpoint. Coordinated checkpointing protocols [19] guarantee that, immediately after a checkpointing session, the *last-node* of every process must be a maximal element; as a result, Case 1 holds for all pairs, thereby achieving the greatest lower bound.

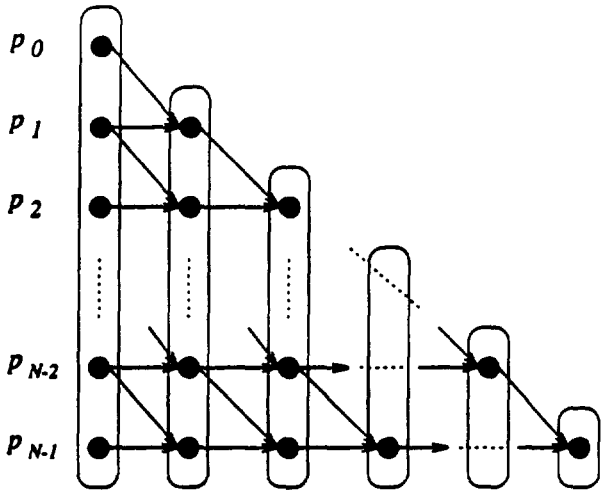


Figure 2.13: G_N^*: The checkpoint graph with $N(N+1)/2$ nongarbage checkpoints.

2.3 Optimal Message Log Reclamation

As described in Section 1.2, some protocols require message logging to record the in-transit messages. Message logs thus constitute another source of space overhead in addition to checkpoints. It should be noted that message logging can have two purposes. If the message logs are used for state reconstruction in the piecewise deterministic model as described in the next section, then both the *message contents* and the *ordinal positions* [30] (or *state interval indices* [33]), i.e., order of processing, are required. If the message logs are used for recording in-transit messages as is considered in this section, then only *message contents* are important because such messages are allowed to arbitrarily interleave with messages from other incoming channels. For our purpose, a message log is nongarbage if and only if it can become an in-transit message with respect to a possible future recovery line. Since our development of the algorithm will be based upon the checkpoint graphs, we first define a ***nongarbage edge*** as follows. Let $(c_{j,y}, c_{i,x})$ denote

the edge representing the relation $c_{j,y} < c_{i,x}$. Given any consistent global checkpoint M, we say "$(c_{j,y}, c_{i,x})$ *intersects* M," if $c_{j,y} < M[j]$ and $M[i] < c_{i,x}$. By the definition of in-transit messages, we say an edge is nongarbage if and only if it can *intersect* a future recovery line. We will demonstrate that the recovery line transformation and decomposition defined in the previous section can also be used to derive the necessary and sufficient condition for identifying all nongarbage edges. More precisely, we prove that an edge is nongarbage if and only if it *intersects* $M^*(\hat{G} - n_i)$ for some $0 \leq i \leq N-1$. All the message logs corresponding to the *garbage edges* can then be garbage-collected.

2.3.1 Recovery line transformation

Given any nongarbage edge (c, c') in G which *intersects* a future recovery line, we will show that (c, c') must also *intersect* $M^*(\hat{G} - W)$ for some $W \subseteq B$, after repeatedly and alternately applying the transformations within an operational session and across a recovery session.

PROPERTY 6 *For any edge $(c_{j,y}, c_{i,x})$ in a checkpoint graph G, if $(c_{j,y}, c_{i,x})$ intersects the recovery line of a potential supergraph G', then $(c_{j,y}, c_{i,x})$ must also intersect the recovery line of an immediate supergraph of G.*

Proof. Suppose $(c_{j,y}, c_{i,x})$ *intersects* $M^*(G') = M_1 \cup M_2$ where $G = (V, E), G' \in \mathcal{G}_s(G)$, $M_1 = M^*(G') \cap V$ and $M_2 = M^*(G') \setminus V$, as shown in Fig. 2.14. We want to show that $(c_{j,y}, c_{i,x})$ *intersects* $M^*(\hat{G} - B_1) = M_1 \cup B_2$, where $B_1 = \{n_i : M^*(G')[i] \in M_1\}$

and $B_2 = \{n_i : M^*(G')[i] \in M_2\}$, which is the recovery line obtained by applying the transformation within an operational session to $M^*(G')$.

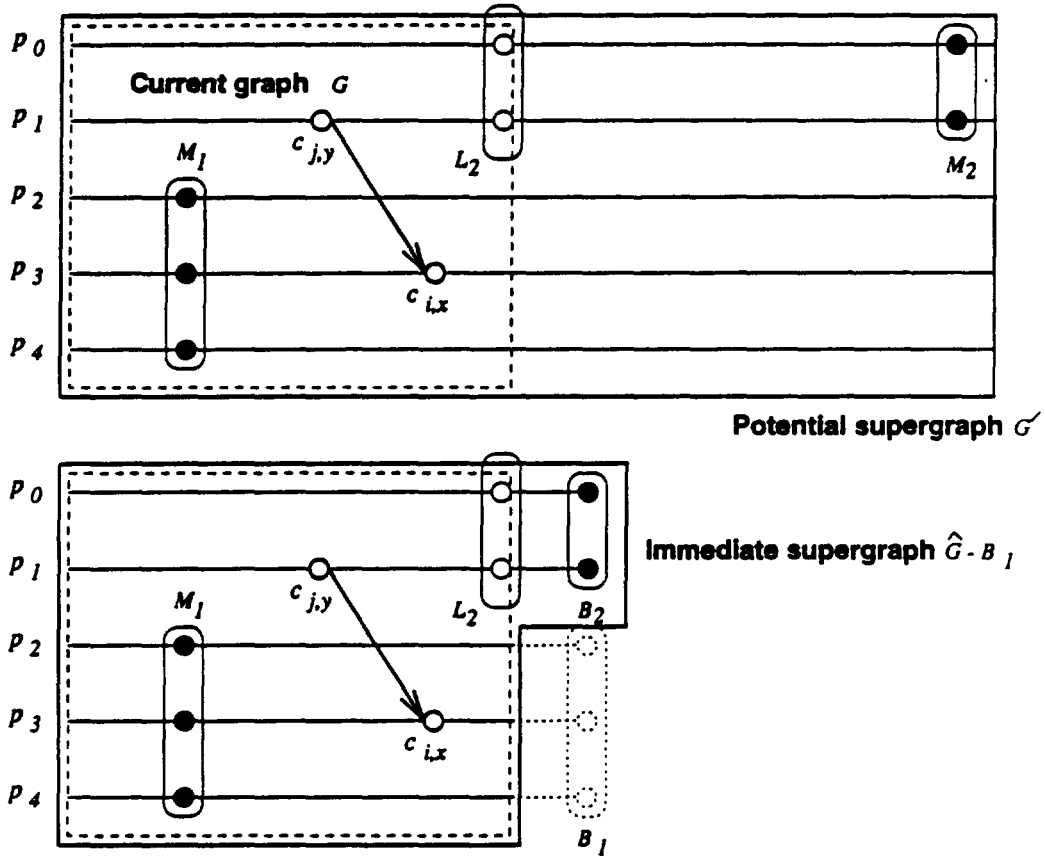


Figure 2.14: Nongarbage edge in the transformation within an operational session.

By definition, $c_{j,y} < M^*(G')[j]$ and $M^*(G')[i] < c_{i,x}$. Since $c_{i,x} \in V$, we must have $M^*(G')[i] \in M_1$ and hence $M^*(\hat{G} - B_1)[i] = M^*(G')[i] < c_{i,x}$. Similarly, if $M^*(G')[j] \in M_1$, then $c_{j,y} < M^*(\hat{G} - B_1)[j]$; otherwise, $M^*(G')[j] \in M_2$ and we must have $c_{j,y} \leq l_j < n_j = M^*(\hat{G} - B_1)[j]$. Therefore, we have shown $(c_{j,y}, c_{i,x})$ *intersects* $M^*(\hat{G} - B_1)$ as required. □

PROPERTY 7 *Let G and G^- denote the checkpoint graphs immediately before and after a recovery session, respectively. For any edge $(c_{j,y}, c_{i,x})$ in G^-, if $(c_{j,y}, c_{i,x})$ intersects the recovery line of an immediate supergraph of G^-, then $(c_{j,y}, c_{i,x})$ must also intersect the recovery line of an immediate supergraph of G.*

Proof. Suppose $(c_{j,y}, c_{i,x})$ *intersects* $M^*(\hat{G}^- - B_1^-) = M_1 \cup M_2 \cup M_3$ as shown in Fig. 2.15, where M_i's were defined in Eq. (2.2) and B_1^- was defined in Eq. (2.4). We want to show that $(c_{j,y}, c_{i,x})$ *intersects* $M^*(\hat{G} - (T_1 \cup B_1)) = M_1 \cup M_2 \cup M_4$, where T_1 was defined in Eq. (2.1) and B_1 was defined in Eq. (2.3), which is the recovery line obtained by applying the transformation across a recovery session to $M^*(\hat{G}^- - B_1^-)$.

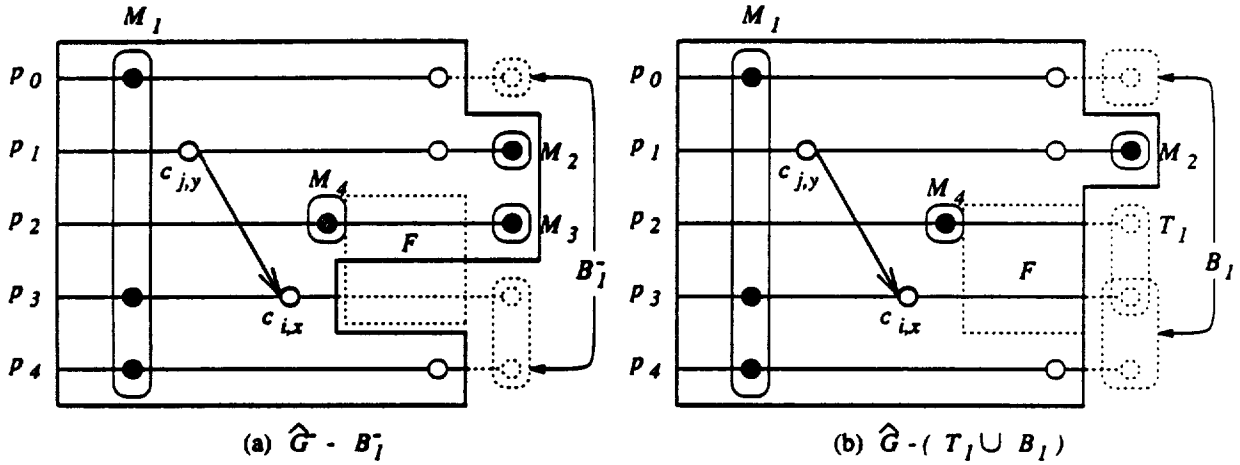


Figure 2.15: Nongarbage edge in the transformation across a recovery session.

Let $M_0^* = M^*(\hat{G} - (T_1 \cup B_1))$ and $M_1^* = M^*(\hat{G}^- - B_1^-)$. By definition, $c_{j,y} < M_1^*[j]$ and $M_1^*[i] < c_{i,x}$. Following the same arguments as in the proof of Property 6, we have $M_0^*[i] = M_1^*[i] < c_{i,x}$ and, if $M_1^*[j] \in M_1 \cup M_2$, $c_{j,y} < M_0^*[j]$. If $M_1^*[j] \in M_3$, we still have $c_{j,y} < M_0^*[j]$ unless $c_{j,y} \in M_4$. Since Rule 2.a guarantees that none of the vertices in M_4

can have any outgoing edge in G^-, $c_{j,y} \notin M_4$ and therefore we have shown that $(c_{j,y}, c_{i,x})$ *intersects* M_0^* as required. □

Combining Properties 6 and 7 and following the proof of Property 3 immediately lead to the following result.

PROPERTY 8 *If an edge $(c_{j,y}, c_{i,x})$ in a checkpoint graph G intersects a future recovery line of G, then $(c_{j,y}, c_{i,x})$ must also intersect the recovery line of an immediate supergraph of G.*

2.3.2 Recovery line decomposition

We will first show that Eq. (2.12) in Property 4 can be expressed in terms of a component-wise minimum operation, and then prove that any edge *intersecting* the recovery line of an immediate supergraph must *intersect* one of the N recovery lines $M^*(\hat{G} - n_i), 0 \leq i \leq N-1$.

LEMMA 7 *Given a checkpoint graph G, $W \subseteq B$ and $W \neq \emptyset$,*

$$M^*(\hat{G} - W)[j] = \min(\bigcup_{n_k \in W} M^*(\hat{G} - n_k)[j]) \text{ for all } 0 \leq j \leq N-1.$$

Proof. For any $0 \leq j \leq N-1$, we consider the set of checkpoints $\{M^*(\hat{G} - n_k)[j] : n_k \in W\}$ which contains all of the checkpoints of p_j in $\bigcup_{n_k \in W} M^*(\hat{G} - n_k)$. Only $\min(\bigcup_{n_k \in W} M^*(\hat{G} - n_k)[j])$ can be a minimal element. From Property 4, $M^*(\hat{G} - W) = \min(\bigcup_{n_k \in W} M^*(\hat{G} - n_k))$. Since $M^*(\hat{G} - W)$ must contain one checkpoint from each process, we have $M^*(\hat{G} - W)[j] = \min(\bigcup_{n_k \in W} M^*(\hat{G} - n_k)[j])$ as required. □

PROPERTY 9 *If an edge $(c_{j,y}, c_{i,x})$ in a checkpoint graph G intersects the recovery line of an immediate supergraph of G, then $(c_{j,y}, c_{i,x})$ must also intersect one of the N recovery lines $M^*(\hat{G} - n_k), 0 \leq k \leq N-1$.*

Proof. Suppose that $(c_{j,y}, c_{i,x})$ *does not intersect* any $M^*(\hat{G} - n_k)$. Then, for any $0 \leq k \leq N-1$, either (1) $c_{i,x} \leq M^*(\hat{G} - n_k)[i]$ or (2) $M^*(\hat{G} - n_k)[j] \leq c_{j,y}$. For any immediate supergraph $\hat{G} - W$ of G, if case (1) is true for all $n_k \in W$, then

$$c_{i,x} \leq \min(\bigcup_{n_k \in W} M^*(\hat{G} - n_k)[i]) = M^*(\hat{G} - W)[i]$$

by Lemma 7; if case (2) is true for some $n_l \in W$, then

$$M^*(\hat{G} - W)[j] = \min(\bigcup_{n_k \in W} M^*(\hat{G} - n_k)[j]) \leq M^*(\hat{G} - n_l)[j] \leq c_{j,y}.$$

In either case, $(c_{j,y}, c_{i,x})$ does not *intersect* $M^*(\hat{G} - W)$. Therefore, if $(c_{j,y}, c_{i,x})$ *intersects* $M^*(\hat{G} - W)$, then $(c_{j,y}, c_{i,x})$ must *intersect* $M^*(\hat{G} - n_k)$ for some $0 \leq k \leq N-1$. □

Combining Properties 8 and 9, we now present the necessary and sufficient condition for identifying all nongarbage edges.

THEOREM 4 *An edge $(c_{j,y}, c_{i,x})$ in a checkpoint graph G is nongarbage if and only if $(c_{j,y}, c_{i,x})$ intersects $M^*(\hat{G} - n_k)$ for some $0 \leq k \leq N-1$.*

Proof. If $(c_{j,y}, c_{i,x})$ is nongarbage, $(c_{j,y}, c_{i,x})$ must *intersect* a future recovery line of G by definition. From Property 8, $(c_{j,y}, c_{i,x})$ must *intersect* the recovery line of an immediate supergraph of G. From Property 9, $(c_{j,y}, c_{i,x})$ must *intersect* one of the N recovery lines

$M^*(\hat{G} - n_k), 0 \leq k \leq N-1$. Conversely, if $(c_{j,y}, c_{i,x})$ *intersects* any $M^*(\hat{G} - n_k)$, then $(c_{j,y}, c_{i,x})$ is nongarbage because $M^*(\hat{G} - n_k)$ is a possible future recovery line of G. □

Theorem 4 also leads to an optimal message log reclamation algorithm for finding all nongarbage message logs: first compute the N recovery lines $M^*(\hat{G} - n_k)$, $0 \leq k \leq N-1$; only those message logs with their corresponding edges *intersecting* any of the N recovery lines are nongarbage. In Fig. 2.16, the edge (E, F) *intersects* $M^*(\hat{G} - n_0)$, (G, H) *intersects* $M^*(\hat{G} - n_4)$ and none of the edges *intersects* $M^*(\hat{G} - n_1)$, $M^*(\hat{G} - n_2)$ or $M^*(\hat{G} - n_3)$. Therefore, while all of the edges in Fig. 2.16(f) are nonobsolete, only those message logs corresponding to (E, F) and (G, H) need to be retained.

Figure 2.17 shows an example for analyzing the algorithm complexity. The rollback propagation algorithm is applied to the checkpoint graph shown in Fig. 2.17(a), and (b)-(d) illustrate the steps for finding the recovery line. Since all of the visited edges can be removed as shown in the figure, the algorithm is of time complexity $\mathbf{O}(|E|)$. The nongarbage edges can be identified as the remaining incoming edges of the marked checkpoints. Since the additional complexity of scanning through the marked checkpoints is no greater than $\mathbf{O}(|E|)$, our optimal garbage collection algorithm for identifying all nongarbage edges as well as nongarbage checkpoints is of complexity $\mathbf{O}(N|E|)$.

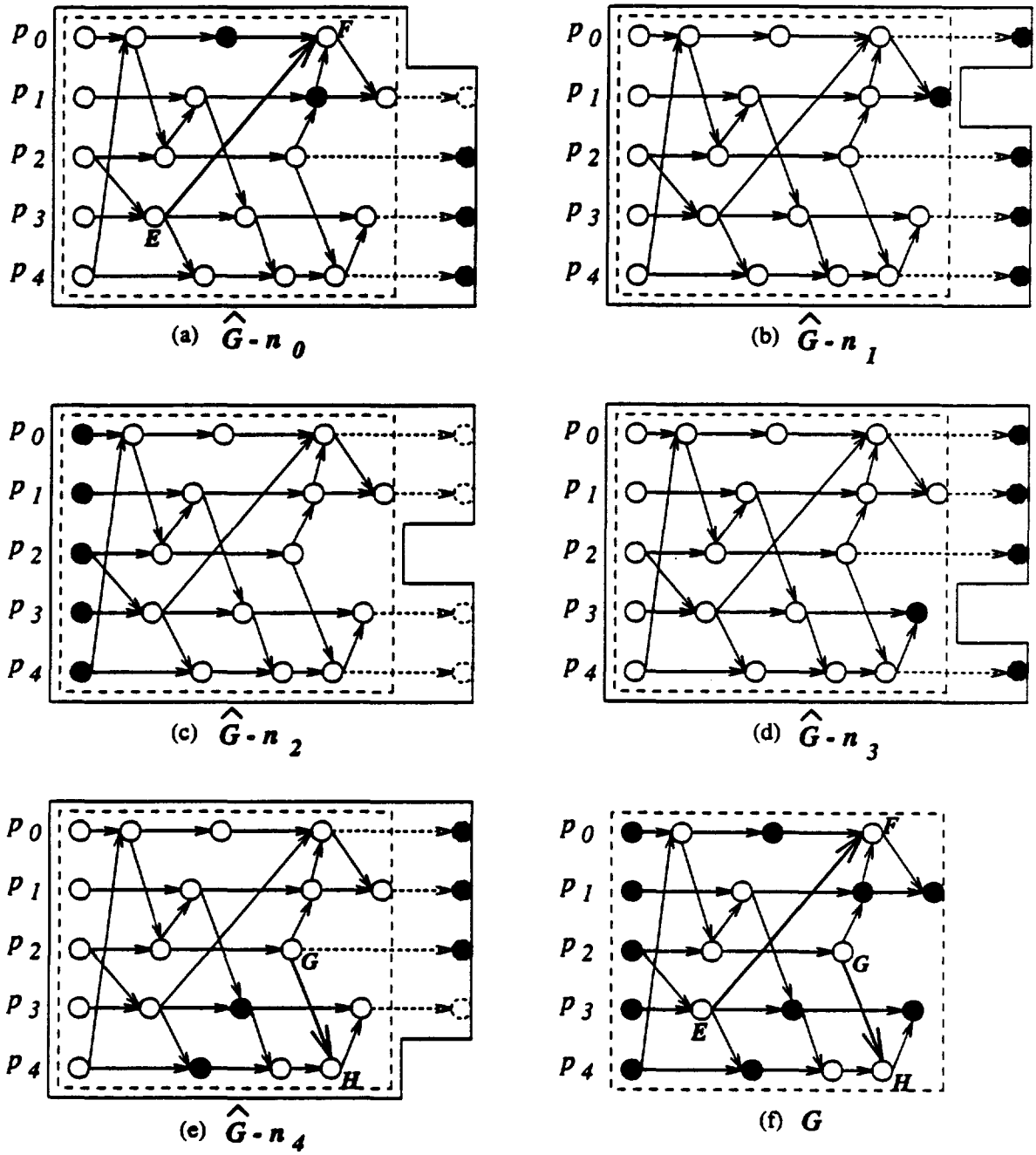


Figure 2.16: Example execution of the optimal garbage collection algorithm.

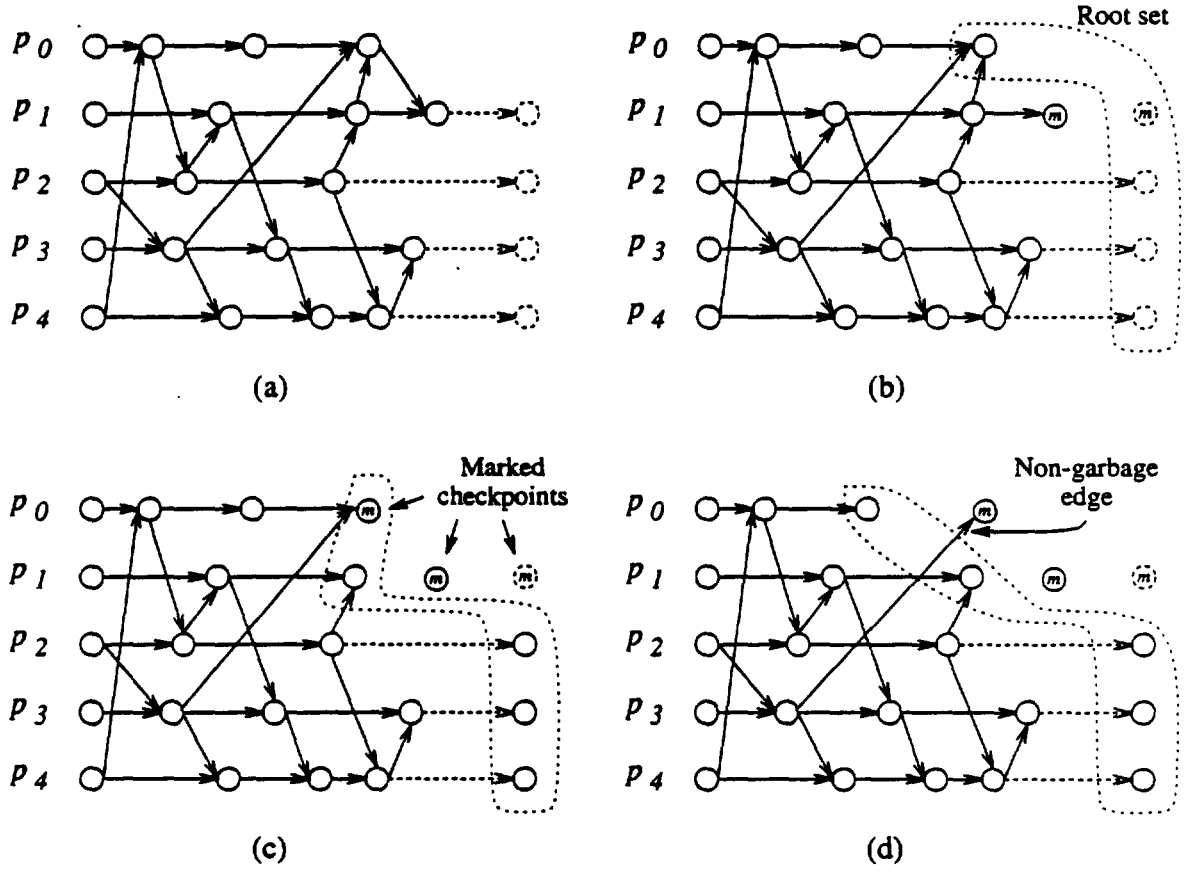


Figure 2.17: Example for identifying nongarbage checkpoints and edges. (For the checkpoint graph in (a), (b)-(d) illustrate the step-by-step execution of the rollback propagation algorithm.)

3. RECOVERY LINE PROGRESSION

Having developed a garbage collection algorithm to minimize the space overhead for uncoordinated checkpointing, we next address the recovery line progression problem. Traditionally, uncoordinated checkpointing, coordinated checkpointing, and the log-based approach have been considered three different approaches, each with its own advantages and disadvantages as described in Chapter 1. In this chapter, we adopt a different viewpoint and present a unifying framework for all three approaches. Our framework is based on uncoordinated checkpointing, because it is the most general approach in terms of process autonomy and the assumptions about program behavior. The price paid for such generality is the potential domino effect. We then consider *checkpoint coordination* and *exploiting piecewise determinism* as two mechanisms for eliminating the domino effect by inserting additional checkpoints. The concept of lazy checkpoint coordination is introduced to allow sacrificing a varying degree of process autonomy in exchange for the guarantee of recovery line progression. The notion of logical checkpoints is introduced to interpret message logging in the piecewise deterministic model as providing additional checkpoints available for the processes to roll back to, thereby reducing rollback propagation. In such a unifying framework, all three approaches can be integrated together and benefit from the optimal garbage collection algorithm of Chapter 2. At the end of

this chapter, we also describe a message reordering approach to reducing rollback propagation without inserting any extra checkpoints. For systems in which messages can be reordered without affecting program correctness, such a technique can be incorporated as an additional mechanism to further advance the recovery lines.

3.1 Lazy Checkpoint Coordination

3.1.1 Communication-induced checkpoint coordination

The basic concept of lazy checkpoint coordination is to insert extra *induced checkpoints* based on the communication history, in addition to the *basic checkpoints* initiated independently by each process, to ensure that a new consistent set of checkpoints will be formed periodically to advance the recovery line. Figure 3.1(a) illustrates a checkpoint and communication pattern with the domino effect. A straightforward way of avoiding such possibly unbounded rollback propagation is to perform traditional *eager checkpoint coordination* as shown in Fig. 3.1(b), where $b_{i,x}$ denotes the xth basic checkpoint of p_i. Whenever a process takes a basic checkpoint, a *coordination message* (dotted line) is broadcast to request the cooperation in making a consistent set of checkpoints [11]. 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 always has $\mathcal{R} = N - 1$, and the $N - 1$ coordination messages per checkpoint session constitute additional overhead.

The large overhead of eager checkpoint coordination results from its pessimistic nature. More specifically, when p_1 in Fig. 3.1(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. To guarantee that $b_{1,1}$ 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 [21] (Fig. 3.1(c)).[1] From another point of view, such a scheme "lazily" delays the broadcast of the coordination messages and implicitly piggybacks them on future normal messages [20]. 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. 3.1(c). To further reduce the overhead, we can perform even "lazier" coordination

[1]The motivation for lazy checkpoint coordination is similar to the concepts behind the *lazy release consistency* in distributed shared memory [51] and the *lazy message cancellation* in optimistic distributed simulation systems [52].

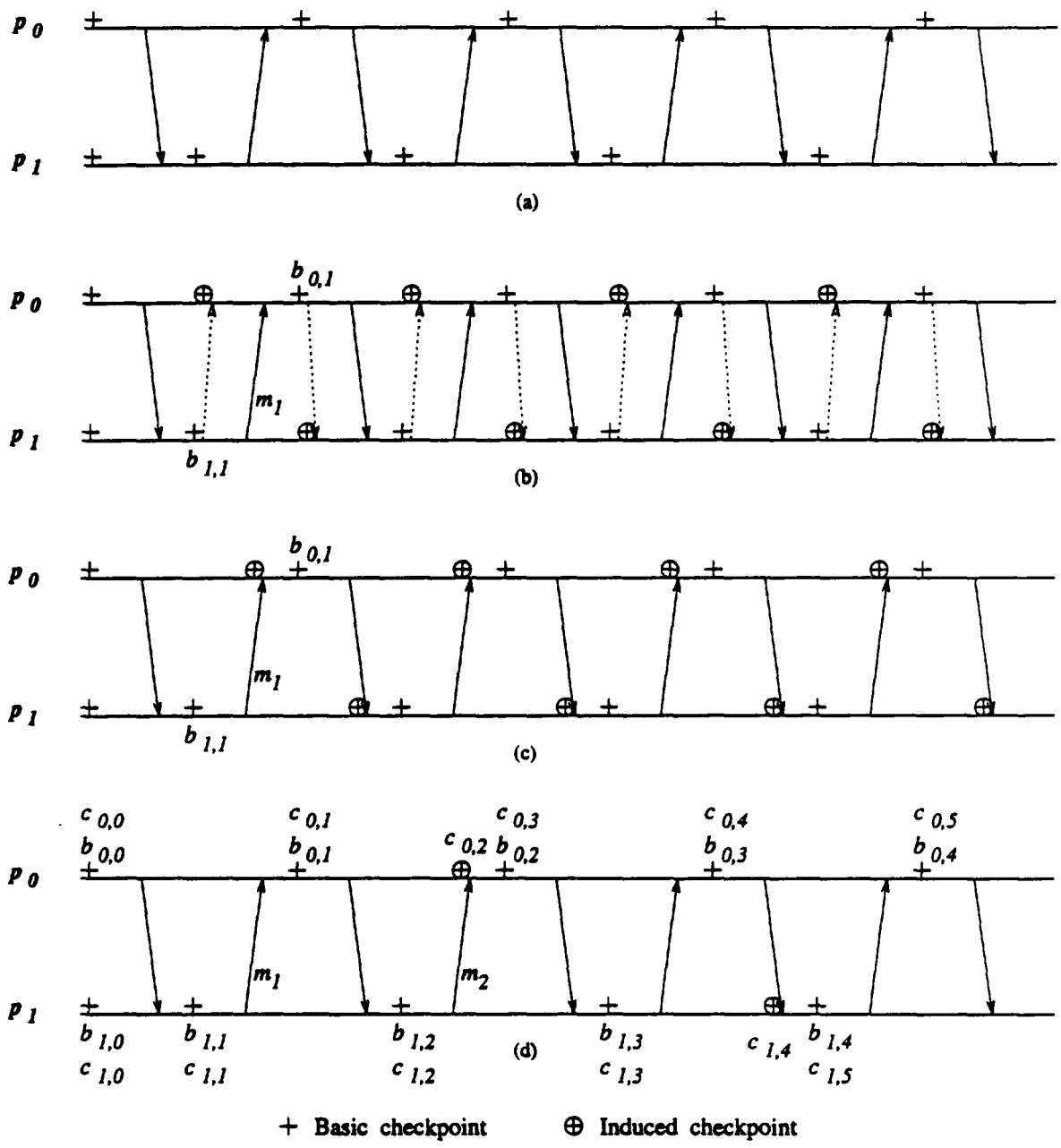


Figure 3.1: Communication-induced checkpointing. (a) checkpoint and communication pattern; (b) eager checkpoint coordination; lazy checkpoint coordination with (c) laziness = 1 and (d) laziness = 2.

by enforcing the consistency only between checkpoints $c_{0,nZ}$ and $c_{1,nZ}$ where Z is called the *laziness* and n is an integer. Figure 3.1(d) shows the case of $Z = 2$. No checkpoint is induced until message m_2 indicates the inconsistency between $b_{1,2}$ and $b_{0,2}$. The number of induced checkpoints is reduced from 8 to 2 at the cost of potentially larger rollback distance. We will show in the next section that, while the worst-case induction ratio for $Z = 1$ is of order N (the number of processes), the upper bound on the induction ratio for $Z \geq 2$ is related to the maximum ratio of the lengths of any two basic checkpoint intervals and is independent of N.

Lazy checkpoint coordination can be implemented as follows. 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, then it must be processed by p_i after $c_{i,lZ}$ is induced. Notice that the induced checkpoint $c_{i,lZ}$ can be referred to as any checkpoint $c_{i,w}$ with $x < w \leq lZ$. Since our approach is to incorporate lazy checkpoint coordination into an uncoordinated checkpointing protocol (which corresponds to $Z = \infty$) as a mechanism for bounding rollback propagation, the optimal garbage collection algorithm remains useful in reducing the space overhead for such a domino-free recovery protocol.

3.1.2 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. Second, 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 that 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[2] $\mathcal{P}_0$ as follows. 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 $t(c^{\mathcal{P}_0}_{j,nZ}) \approx t(c^{\mathcal{P}_0}_{i,nZ})$. Figure 3.2(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 of 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 $\#nZ$, e.g., $c^{\mathcal{P}}_{i,nZ}$, is taken, p_i has to take Z consecutive basic checkpoints by itself to reach $c^{\mathcal{P}}_{i,nZ}$, as stated in Property 10.

PROPERTY 10 *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.*

[2]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.*

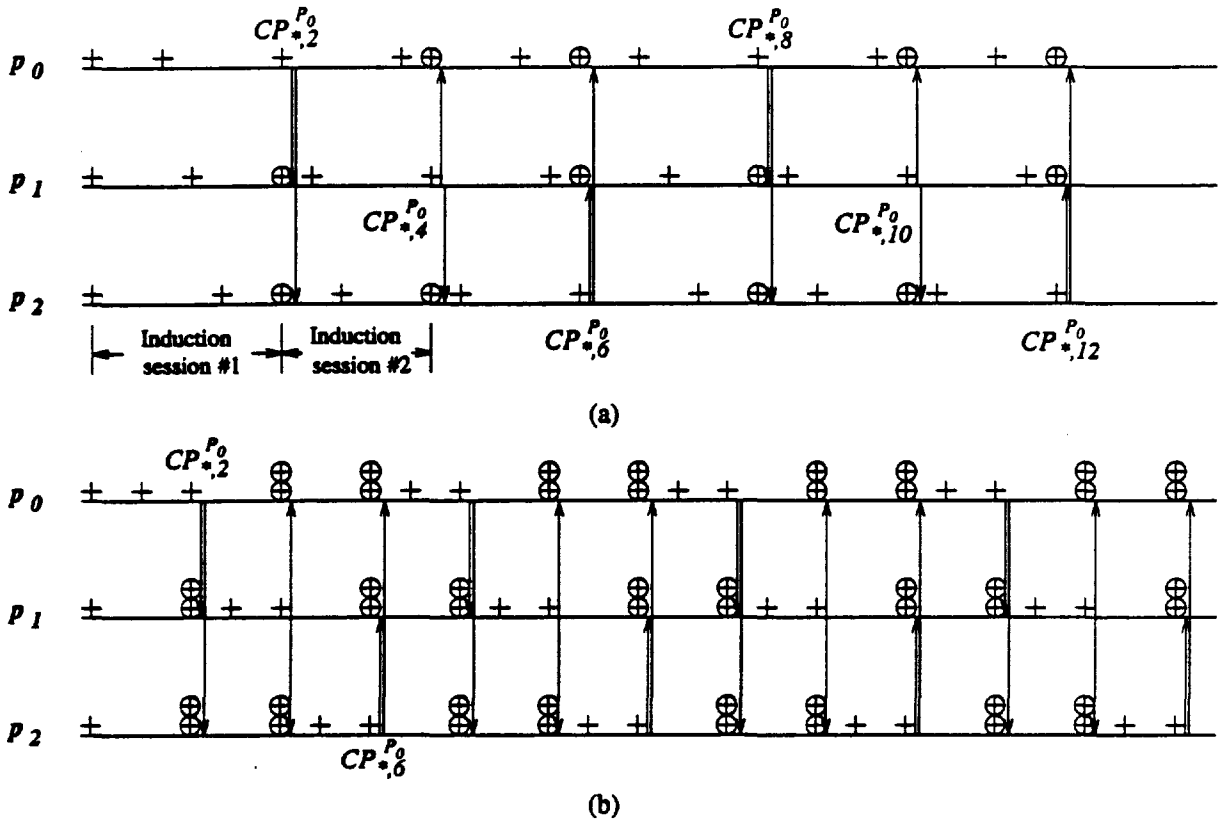


Figure 3.2: (a) Worst-case communication pattern (b) worst-case checkpoint and communication pattern.

We show in the next property that, given a basic checkpoint pattern, $\mathcal{P}_0$ has the earliest $c^{\mathcal{P}}_{*,nZ}$ for any positive integer n.

PROPERTY 11 *Given a basic checkpoint pattern, we have* $t(c^{\mathcal{P}_0}_{*,nZ}) \leq t(c^{\mathcal{P}}_{*,nZ})$ *for arbitrary communication pattern* $\mathcal{P}$ *and any positive integer* n.

Proof. The proof is given by induction on n. Since there can not be any induced checkpoint before $t(c^{\mathcal{P}}_{*,Z})$ for any $\mathcal{P}$, $t(c^{\mathcal{P}}_{*,Z})$ depends only on the progress of taking basic checkpoints. Therefore, $t(c^{\mathcal{P}_0}_{*,Z}) = t(c^{\mathcal{P}}_{*,Z})$ and the case $n = 1$ is true. For the case $n = k$, suppose $c^{\mathcal{P}}_{*,kZ} = c^{\mathcal{P}}_{i,kZ}$. All of the Z checkpoints $c^{\mathcal{P}}_{i,l}$ with $(k-1)Z < l \leq kZ$ must be

basic checkpoints by Property 10. Also, $t(c^{\mathcal{P}}_{*,(k-1)Z}) \leq t(c^{\mathcal{P}}_{i,(k-1)Z}) \leq t(c^{\mathcal{P}}_{i,l}) \leq t(c^{\mathcal{P}}_{i,kZ})$ by definition. Suppose that the case $n = k-1$ is true, i.e., $t(c^{\mathcal{P}_0}_{*,(k-1)Z}) \leq t(c^{\mathcal{P}}_{*,(k-1)Z})$. We then have $c^{\mathcal{P}}_{i,kZ} = c^{\mathcal{P}_0}_{i,q}$ where $q \geq kZ$ because $t(c^{\mathcal{P}_0}_{i,(k-1)Z}) \approx t(c^{\mathcal{P}_0}_{*,(k-1)Z})$ by construction and there are at least Z basic checkpoints of p_i, i.e., the $c^{\mathcal{P}}_{i,l}$'s, between $t(c^{\mathcal{P}_0}_{i,(k-1)Z})$ and $t(c^{\mathcal{P}}_{i,kZ})$. Finally,

$$t(c^{\mathcal{P}_0}_{*,kZ}) \leq t(c^{\mathcal{P}_0}_{i,kZ}) \leq t(c^{\mathcal{P}_0}_{i,q}) = t(c^{\mathcal{P}}_{i,kZ}) = t(c^{\mathcal{P}}_{*,kZ})$$

and we have proved $t(c^{\mathcal{P}_0}_{*,nZ}) \leq t(c^{\mathcal{P}}_{*,nZ})$ for all positive integer n. □

It follows from Property 11 that $\mathcal{P}_0$ 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 hence we have the following property.

PROPERTY 12 *Given a basic checkpoint pattern, $\mathcal{P}_0$ is the worst-case communication pattern resulting in the largest induction ratio.*

Property 12 states that, for the worst-case analysis of induction ratio, we have to consider only the communication pattern $\mathcal{P}_0$ for each basic checkpoint pattern. Since every $\mathcal{P}_0$ has well-defined induction sessions as shown in Fig. 3.2, the derivations can be greatly simplified.

From Property 10, at least Z basic checkpoints are needed to induce at most $N-1$ checkpoints and thus we have an upper bound on the induction ratio

$$\mathcal{R} \leq \frac{N-1}{Z} \tag{3.1}$$

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. 3.2(b). (The stacked checkpoints indicate that each dummy checkpoint $c_{i,2n-1}^{\mathcal{P}_0}$ is exactly the induced checkpoint $c_{i,2n}^{\mathcal{P}_0}$.)

The upper bound in Eq. (3.1) was derived under no constraints on the checkpoint and communication pattern. Since it is $\mathbf{O}(N)$, 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. 3.2 reveals that the situation in (b) which results in the worst-case induction ratio is less likely to happen for applications in which 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 $t(c_{*,2}^{\mathcal{P}_0})$ and $t(c_{*,6}^{\mathcal{P}_0})$. We will show that under the following constraints, which are usually 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. 3.1(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 lengths 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.) Therefore, $Q = \mathbf{O}(1)$.

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 that $Z \cdot L \gg \lceil Q \rceil$.

From Property 10, each induction session must contain Z consecutive basic checkpoints and hence at least $Z-1$ basic checkpoint intervals at some process. 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 $M \cdot (N-1)$ where $M \leq \lceil Q \rceil = \mathbf{O}(1)$ and the number of basic checkpoints is at least N, the upper bound on the induction ratio is independent of N.

THEOREM 5 *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 lengths of any two basic checkpoint intervals.*

Proof. Again we have to consider only $\mathcal{P}_0$ 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. (3.1), $\mathcal{R} \leq \frac{N-1}{Z} < N < M \leq \lceil Q \rceil$.

(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 $Z < Q+1$, then $Z \cdot L \gg \lceil Q \rceil$ in Constraint-2 implies

$L \gg \lceil Q \rceil$. Since $M \leq \lceil Q \rceil$, we have $L/M \gg 1$; thus $L_M \gg 1$ 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 $c^{\mathcal{P}_0}_{i,nZ} = c^{\mathcal{P}_0}_{*,nZ}$ must contribute Z basic checkpoints. 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 $c^{\mathcal{P}_0}_{*,nZ}$ 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,

$$\begin{aligned} B &\geq L_M \cdot (M \cdot Z + (N-M) \cdot \lfloor \frac{M \cdot (Z-1)}{Q} \rfloor) \text{ and} \\ \mathcal{R} &= \frac{I}{B} \leq \frac{M \cdot (N-1)}{M \cdot Z + (N-M) \cdot \lfloor \frac{M\cdot(Z-1)}{Q} \rfloor}. \end{aligned} \tag{3.2}$$

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

$$\mathcal{R} < \frac{M \cdot (N-1)}{M + (N-M)} < M \leq \lceil Q \rceil \tag{3.3}$$

as required. □

Combining Eq. (3.1) (for $Z = 1$ and Case(a)) and Eq. (3.2), we define the refined upper bound on the induction ratio R, 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.

3.1.3 Experimental results

Table 3.1 gives the parameters of the four Chare Kernel programs used in the trace-driven simulation with lazy checkpoint coordination. The predetermined minimum basic checkpoint interval is chosen to be 120 sec. A variable *Next_CP_Time* is initialized to 120 sec. Each process checks its local clock after processing every 100 messages. If the clock time exceeds *Next_CP_Time*, then a basic checkpoint is inserted, and *Next_CP_Time* is incremented by 120 sec. The resulting average basic checkpoint interval (CPI) for each program is listed in Table 3.1. Before processing a new message, each process also checks if it has to take an induced checkpoint, as described in Section 3.1. All reported numbers are averaged over five runs.

Table 3.1: 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 number of basic checkpoints per processor	12.6	11.8	18.0	10.5
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%

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 3.1. 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. Figure 3.3 plots the Q-bounds against the worst-case ratios computed from Eq. (3.1) and the actual induction ratios (the "Result" curve) obtained from the trace-driven simulation 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 [21] can significantly reduce the extra checkpoint overhead.

Figure 3.4 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. Figures 3.3 and 3.4 illustrate that lazy checkpoint coordination provides a flexible trade-off between coordination overhead and recovery efficiency.

3.2 Exploiting Piecewise Determinism

As described in Section 1.2, when p_j in Fig. 3.5 rolls back to $c_{j,y}$, p_i is also required to roll back to undo the effect of message m because the potential nondeterministic events preceding the sending of m, for example, the event e_1, may invalidate m after the rollback. However, if the event e_1 can be detected and recorded, it can then be replayed after the rollback to reconstruct the state from which message m was originally sent (under the fail-stop assumption). Since the exact copy of m is guaranteed to be resent during

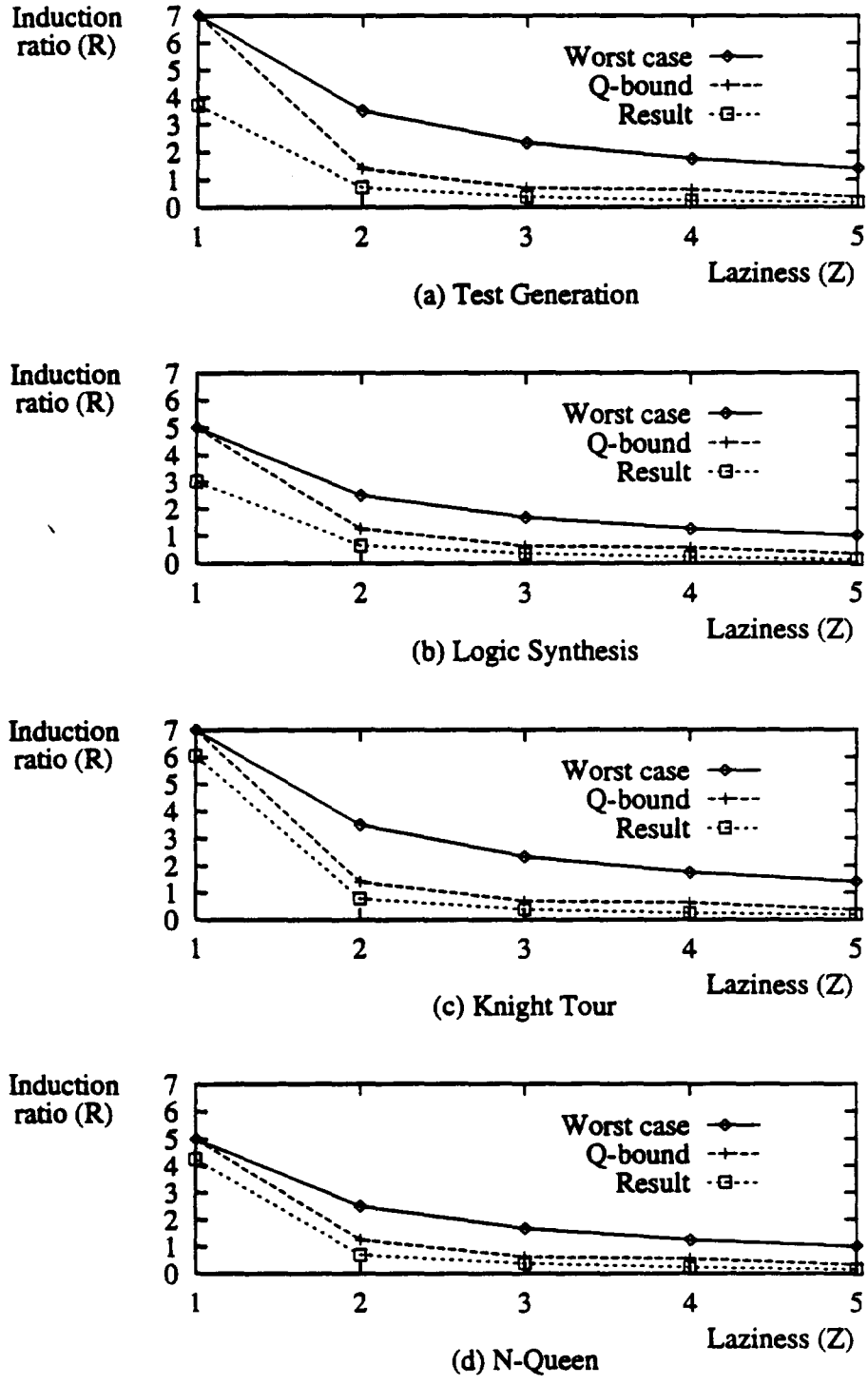


Figure 3.3: Checkpoint coordination overhead (induction ratio) as a function of laziness.

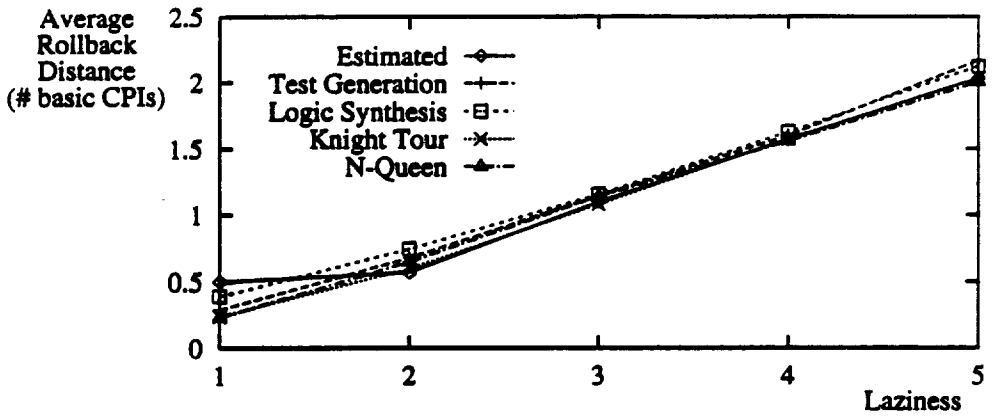


Figure 3.4: Average rollback distance as a function of laziness.

reexecution, the state of p_i, which depends on the processing of m, remains valid and does not have to be rolled back. Motivated by the above observation, the *log-based approach* [26] assumes the *piecewise deterministic (PWD) model* in which the process execution is viewed as consisting of a number of *state intervals* with completely deterministic execution, each started by a detectable and recordable nondeterministic event, for example, processing a new message. It has been shown that under the PWD assumption, additional event logging allows deterministic state reconstruction to effectively eliminate the domino effect [26, 33]. Because the concept of state consistency in the PWD model is fundamentally different from the one described in Section 1.2, the log-based approach has traditionally been presented using a different dependency model (as described in the next chapter).

We first present a unified dependency model for both PWD and non-PWD scenarios. Our approach is to introduce the notion of logical checkpoints which allows the *happened*

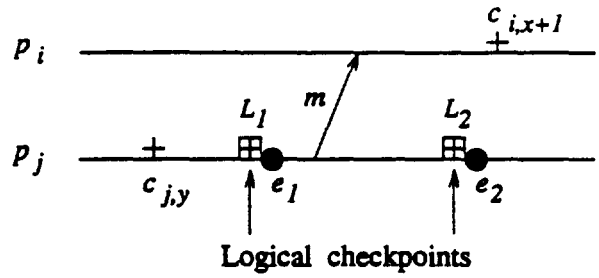


Figure 3.5: Nondeterministic events and logical checkpoints.

before dependency model to be applied to systems with piecewise determinism as well. Suppose e_1 and e_2 are the only nondeterministic events of p_j in Fig. 3.5. We call $c_{j,y}$ a *physical checkpoint* which allows the process state to be restored by simply loading the saved state on stable storage back to memory. With the PWD assumption, restarting the execution from $c_{j,y}$ also guarantees that the state up to the point immediately before the next nondeterministic event e_1 can be reconstructed, effectively placing a *logical checkpoint* L_1 before e_1. In addition, if event e_1 is logged and can be replayed, then p_j's capability of state reconstruction equivalently places another logical checkpoint L_2 immediately before e_2. Therefore, while p_j *physically rolls back* to $c_{j,y}$, it *logically rolls back* to L_2 and does not *unsend* m. In other words, while $c_{j,y}$ and $c_{i,x+1}$ are inconsistent, L_2 and $c_{i,x+1}$ are not ordered by the *happened before* relation and thus form a consistent set of checkpoints.

It becomes clear that the PWD assumption allows the use of event logging to increase the number of available (logical) checkpoints, thereby reducing rollback propagation and avoiding domino effects. However, the assumption that every nondeterministic event in the entire execution is detectable, recordable and replayable may not be valid for many

applications [53, 54]. For example, replaying real-time clock values, sensor readings or external resource status may not be meaningful. Therefore, instead of viewing the PWD assumption as a restriction imposed upon the program behavior, we consider exploiting the PWD model as a mechanism for reducing rollback propagation. More specifically, we use uncoordinated checkpointing as the basic scheme and exploit the PWD model whenever it is valid in order to effectively advance the recovery line.

Figure 3.6 gives an example. In Fig. 3.6(a), no piecewise determinism is exploited and the domino effect forces the global recovery line to stay at the very beginning of process execution. If piecewise determinism can be fully exploited, as shown in Fig. 3.6(b), then we have an additional logical checkpoint before every nondeterministic event. Figure 3.6(c) shows a situation in which the PWD assumption is not valid for the parts of the execution indicated by the shaded bars, and hence the logical checkpoints in those regions are unavailable. Note that once a process turns off the PWD mode, it cannot resume PWD execution until the next physical checkpoint because the state reconstruction process for current checkpoint interval has been interrupted.

Figure 3.7(a) gives the corresponding (logical) checkpoint graph and (b)-(f) show the N recovery lines produced by the PCSR algorithm of Section 2.1.4. We note that since the logical checkpoint L_0 in (b) is nongarbage, the physical checkpoint c_0 and the message log of m_0 (Fig. 3.6(c)) are nongarbage, and m_0 must be the first new message to be processed after p_0 restarts from c_0. In contrast, the two dark edges in (f) are identified as nongarbage edges by algorithm of Section 2.3, which means that the contents of the

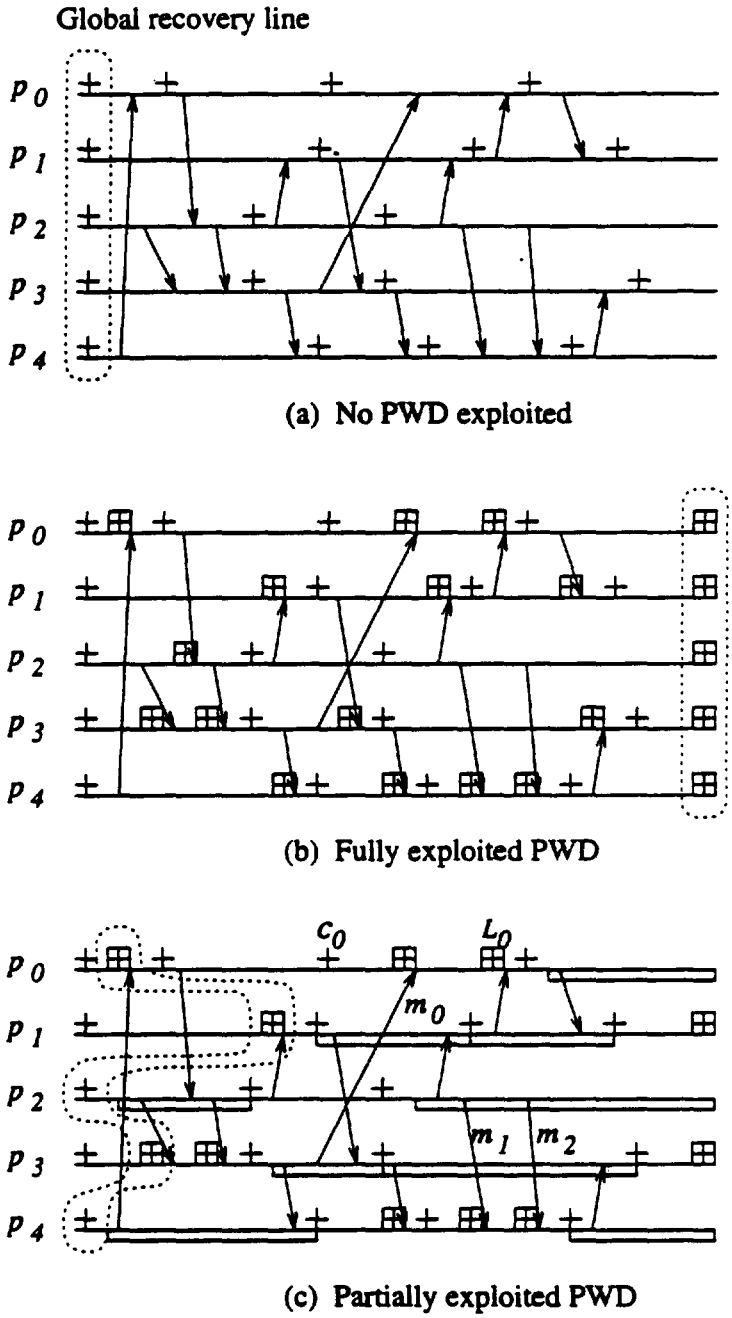


Figure 3.6: Piecewise determinism and the availability of logical checkpoints. (The shaded bars indicate those parts of process execution which do not satisfy the PWD assumption.)

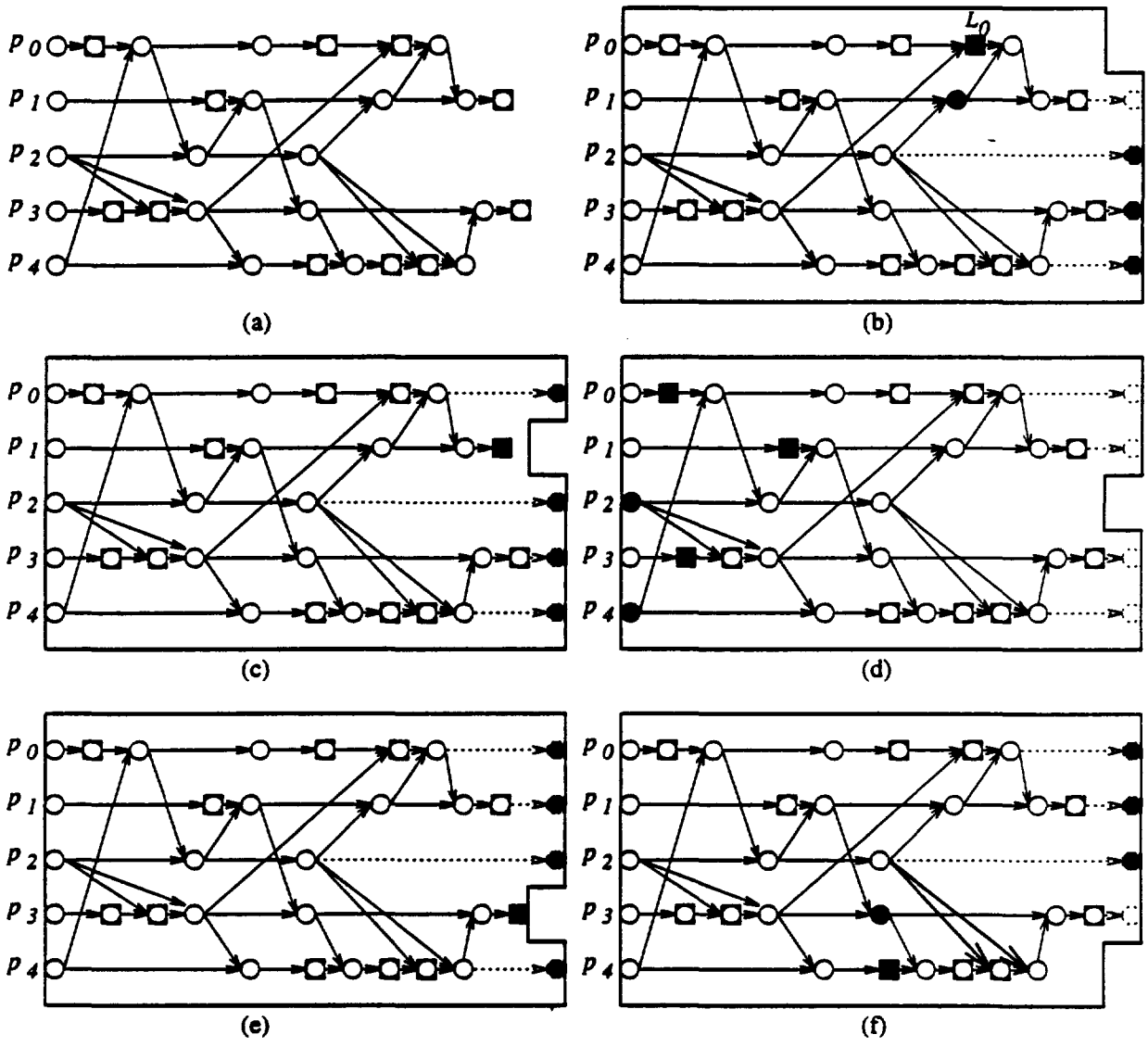


Figure 3.7: Optimal garbage collection for partially exploited piecewise deterministic model. (For the checkpoint graph in (a), (b)-(f) are the N immediate supergraphs used in the PCSR algorithm.)

message logs of m_1 and m_2 are nongarbage but the original order of processing can be discarded.

3.3 Scheduling Message Processing

3.3.1 Message prioritization

Most checkpointing and recovery techniques have assumed that the *communication pattern* is determined by program behavior and is not otherwise controllable; hence, the only way of reducing rollback propagation is to change the *checkpoint pattern* by inserting additional checkpoints. However, in a message-driven system such as the Chare Kernel [55], the communication pattern can often be determined by the run-time support system in a user-transparent way as well as by program behavior. Since the order in which the messages arrive at the receiver can not be assumed, changing the order of message processing will typically not affect program correctness. We next describe how messages can be prioritized according to the checkpointing and communication history in order to control the communication pattern to reduce rollback propagation [56]. Essentially, our message scheduling algorithm is based on the following two concepts: dependency-redundant messages and message aging.

A message m is a *dependency-redundant message* if its immediate processing will result in a dependency that has already been implied in the transitive closure of the current checkpoint graph. Since the recovery line is determined by the transitive closure,

processing a dependency-redundant message will not cause any additional rollback propagation. For example, suppose messages m_0 and m_1 in Fig. 3.8(a) have been processed. Then message m_2 becomes a dependency-redundant message because its corresponding dependency $A < E$ is implied through $A < B < C < D < E$, as shown in (b). Similarly, m_3 is a dependency-redundant message because $B < F$ is implied through $B < C < F$.

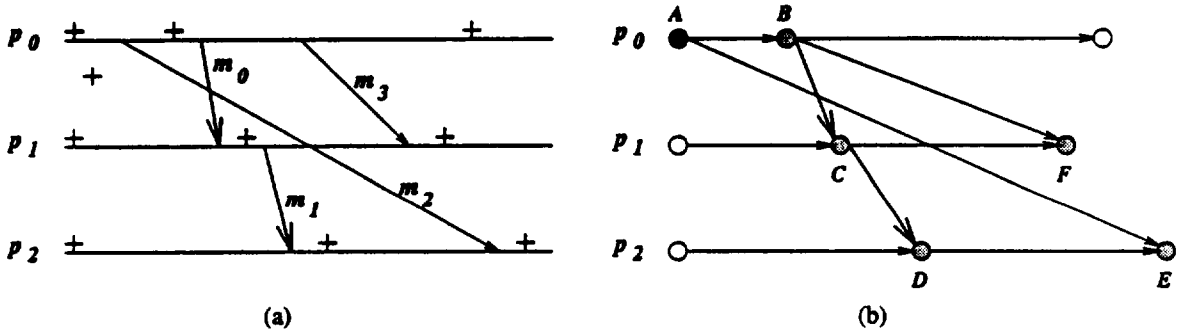


Figure 3.8: Dependency-redundant messages. (a) Checkpoint and communication pattern; (b) checkpoint graph.

The concept of *message aging* is motivated by the communication-induced checkpointing schemes [7, 57, 58], in which a checkpoint is inserted immediately after every message is sent. Since such schemes guarantee that the rollback of any process will not *unsend* any messages, rollback propagation and hence the domino effect are completely eliminated. However, experimental results have shown that the major disadvantages of such schemes are the uncontrollability of the checkpoint frequency [59] and the possibly excessive number of induced checkpoints [60]. Instead of forcing the message senders to take additional checkpoints to ensure that every message sent is processed after its sender takes the next checkpoint, *message aging* encourages the receivers to delay the processing of each message m until its sender passes the next checkpoint. (Such a message m will

be called an *aged message.*) The advantage is that the number of checkpoints and the checkpoint frequency can be independent of the communication patterns; the potential disadvantage is that either the delayed processing might result in run-time overhead, or some processes may be forced to process nonaged messages and hence the system would no longer be free of rollback propagation.

3.3.2 Implementation

For the receiver to detect aged messages, an additional piece of information has to be piggybacked on each message: the time to the next checkpoint of the sender when the message is sent. The receiver can then properly manage its message queue based on this information.

Instead of keeping messages from different processes in the same queue, each process maintains an array of subqueues, one for each process, and a highest-priority *safe queue* for holding dependency-redundant messages and aged messages. Three additional data structures are needed for proper queue management:

1. *Last_Update_Time* records the time at which the most recent update of the time-to-next-checkpoint information was completed. It is needed for the aging operation described later.

2. *Last_Known_CP_Num*[N] is an array, with one entry for each process, recording the most recent checkpoint interval number of every process that is known to the local process based on the communication history.

3. *Last_Processed_CP_Num*[N] is an array recording the highest checkpoint interval number of the processed messages from each process. It is used for identifying dependency-redundant messages.

The updates of the time-to-next-checkpoint information and priorities take place only when a new message arrives (*enqueueing*) or when the process is about to process the next message (*dequeueing*). The operations performed on the message queue for enqueueing and dequeueing are outlined in Figs. 3.9 and 3.10, respectively. The *aging* operation updates the time-to-next-checkpoint information of the last message in each nonempty subqueue by the amount of the difference between current time and *Last_Update_Time*. If the time to the next checkpoint of a message becomes negative, all of the messages in the same subqueue are moved to the *safe queue*.

```
/* message m from the ith checkpoint interval of p_s arrives at the message
   queue Q on p_r */
perform aging operation on Q;
if (i < Last_Known_CP_Num[p_s])
   add m to safe queue;
else {
   if (i > Last_Known_CP_Num[p_s])
      Last_Known_CP_Num[p_s] = i;
   if (i ≤ Last_Processed_CP_Num[p_s])
      add m to the safe queue;
   else
      add m to subqueue[p_s];
}
```

Figure 3.9: Operations for enqueueing.

```
/* p_r is about to choose a message from queue Q */
perform aging operation on Q;
if (safe queue is nonempty)
   choose a message from the safe queue;
else {
   choose the message m with the smallest time-to-next-checkpoint;
   move the remaining messages in the same subqueue to the safe queue;
/* if m is from the ith checkpoint interval of p_s */
   Last_Processed_CP_Num[p_s] = i.
}
```

Figure 3.10: Operations for dequeueing.

3.3.3 Experimental results

Table 3.2 gives the execution parameters of the four Chare Kernel programs used to obtain the experimental results for the message scheduling algorithm. The checkpoint interval is chosen to be 25 sec. An offset of 1 sec between the corresponding checkpoints of processes p_i and p_{i+1} $(0 \leq i < N-1)$ is introduced to study the effect of checkpoint asynchrony on the rollback distances.

Table 3.2: Execution parameters of the parallel programs.

Chare kernel programs	Matrix Multiplication	Circuit Extraction	Knight Tour	N Queen
Number of processors	4	4	6	6
Number of messages	1216	1315	13118	1622

Figure 3.11 compares the performance of three different message scheduling algorithms: Last-In-First-Out (LIFO), First-In-First-Out (FIFO) and our *PRIoritized Message Process Scheduling* (*PRIMPS*) algorithm. The percentage numbers indicate the

performance degradation of PRIMPS with respect to FIFO. Figure 3.12 compares the average rollback distances in terms of the number of checkpoint intervals for the three algorithms. Figures 3.11 and 3.12 show that our message scheduling algorithm can effectively reduce average rollback distances with little performance degradation. Figure 3.13 illustrates the sensitivity of the three algorithms to the degree of checkpoint asynchrony by varying the offset between corresponding checkpoints for the N-Queen program. It shows that our scheduling algorithm has the additional advantage of being much less sensitive to checkpoint asynchrony than are LIFO and FIFO.

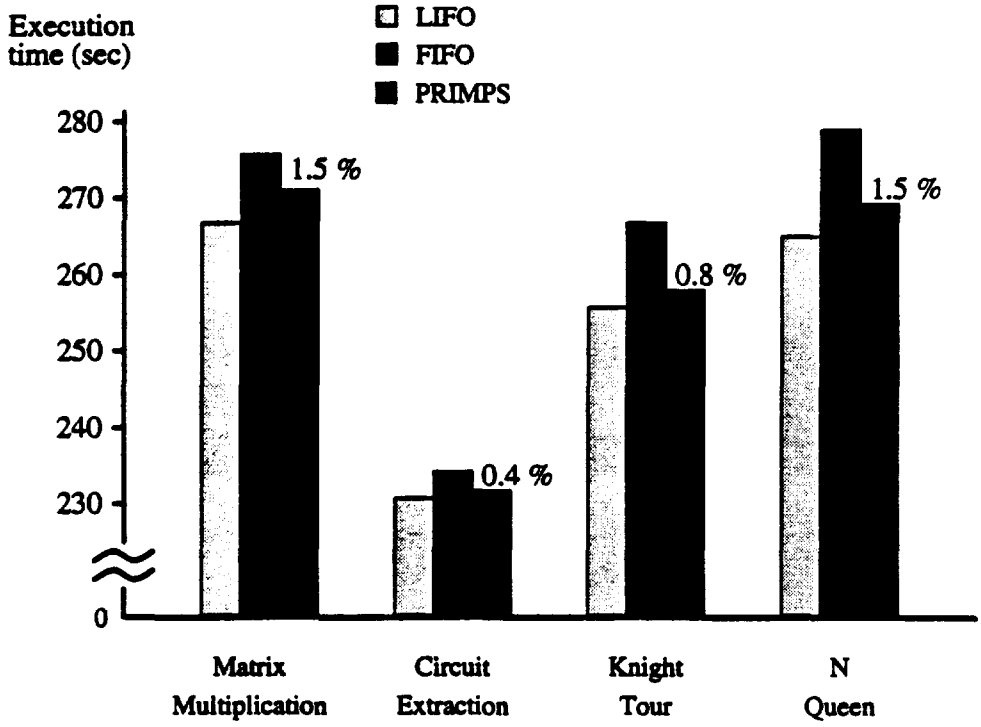


Figure 3.11: Execution times and performance degradation of the message scheduling algorithm.

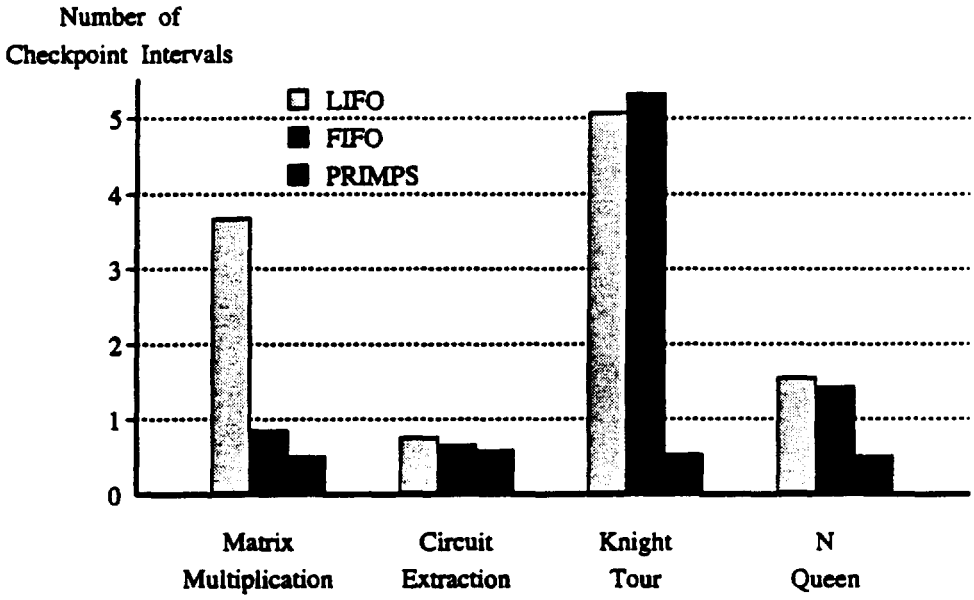


Figure 3.12: Average rollback distances.

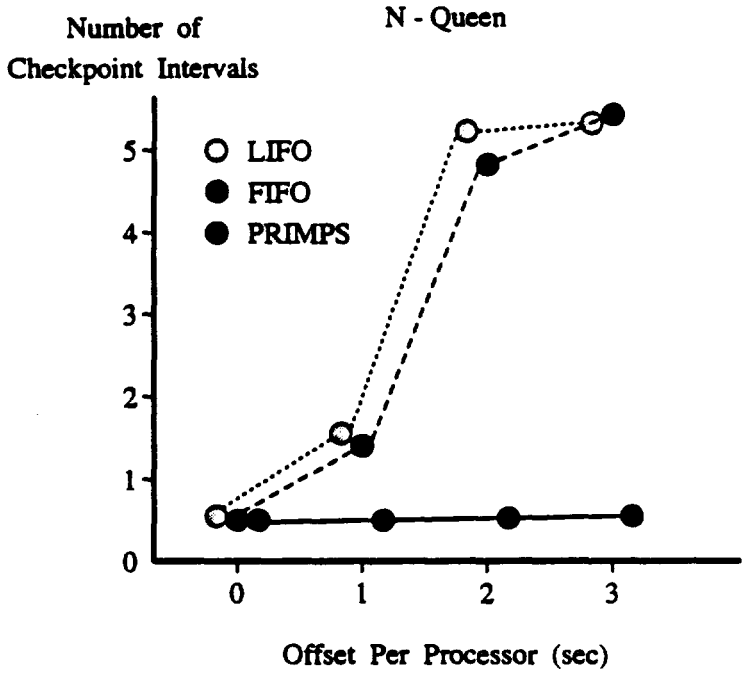


Figure 3.13: Sensitivity of average rollback distances to checkpoint asynchrony for the N-Queen program.

4. RELATED WORK

4.1 Checkpoint Dependency and Interval Dependency

Johnson and Zwaenepoel [33] derived a lattice model for reasoning about recovery in message-passing systems under the assumption of piecewise determinism (PWD). Let (i, x) denote the xth *state interval* of process p_i; they define a dependency relation on the state intervals as follows: (i, x) directly depends on (j, y) if

- $i = j$ and $x = y + 1$; or
- (i, x) is started by a message sent from (j, y).

The transitive closure of the above relation gives the complete state interval dependency. A *system state* consists of N state intervals,[1] one from each process. A *consistent system state* is a system state, of which no two constituent state intervals (i, x) and (j, y) can have (i, x) depending on $(j, y + 1)$ or (j, y) depending on $(i, x + 1)$. A state interval becomes *stable*, i.e., recreatable, when all of the messages processed since its immediate previous physical checkpoint, called its *effective checkpoint*, have been logged. A consistent system state in which all constituent state intervals are stable is called a *recoverable system state.* The recoverable system state with each of its constituent state intervals as advanced as

[1] Johnson and Zwaenepoel originally defined a system state to be an $N \times N$ matrix of which the rows are the transitive dependency vectors. Here we adopt the simplification suggested by Sistla and Welch [31].

possible is called the *maximum recoverable system state.* Johnson and Zwaenepoel have derived the lattice model and proved the uniqueness of the maximum recoverable system state by using the following approach:

Step 1: the set of *system states* forms a lattice $\mathcal{S}$;

Step 2: the set of *consistent system states* forms a sublattice $\mathcal{C}$ of $\mathcal{S}$;

Step 3: the set of *recoverable system states* forms a sublattice $\mathcal{R}$ of $\mathcal{C}$;

Step 4: the *maximum recoverable system state* is the unique maximum in the lattice $\mathcal{R}$.

As described in the previous chapter, by representing every state interval as a logical checkpoint at the end of that interval, the same dependency definition based on the *happened before* relations among checkpoints as used in a non-PWD scenario can still be applied to the logical checkpoints. By referring to the logical checkpoints corresponding to the stable state intervals as *stable logical checkpoints* and the global checkpoints containing only stable logical checkpoints as *stable global checkpoints*, the approach of Johnson and Zwaenepoel can be translated into:

Step 1: the set of *global checkpoints* forms a lattice $\mathcal{S}$;

Step 2: the set of *consistent global checkpoints* forms a sublattice $\mathcal{C}$ of $\mathcal{S}$;

Step 3: the set of *stable consistent global checkpoints* forms a sublattice $\mathcal{R}$ of $\mathcal{C}$;

Step 4: the *recovery line* is the unique maximum in the lattice $\mathcal{R}$.

As a comparison, our derivation of an alternative lattice model as described in Section 1.4 has followed different steps.

Step 1: the set of *logical checkpoints* forms a poset P;

Step 2: the set of *stable logical checkpoints* forms an induced subposet R of P;

Step 3: the set of *stable consistent global checkpoints* is equivalent to the set $\mathcal{M}(R)$ of maximum-sized antichains of R and thus forms a lattice;

Step 4: the *recovery line* is the unique maximal maximum-sized antichain in the lattice $\mathcal{M}(R)$.

We believe our maximum-sized antichain model has several advantages. First, there is a strong intuitive connection between the antichains of the $R_{\mathcal{P}}$ poset based on checkpoint dependencies, and the concept of concurrency and therefore consistency. A consistent global checkpoint must consist of local checkpoints that could have happened simultaneously, and this is precisely captured by the requirement that these local checkpoints be unordered by *happened before*. Johnson and Zwaenepoel's lattice of system states, while perfectly adequate from a formal standpoint, lacks this intuitive force. Furthermore, modelling consistent global checkpoints as maximum-sized antichains enables the use of well-known properties of posets to derive many important results. For example, these properties have played a very important role in our development of the optimal garbage collection algorithm. As another example, our demonstration of the existence of a lattice structure among consistent global checkpoints, and hence the uniqueness of

the recovery line, follows from general theorems about antichains in posets, as contrasted with Johnson and Zwaenepoel's more "low-level" development.

4.2 Checkpoint Graphs and Local System Graphs

Bhargava and Lian [5] consider the same uncoordinated checkpointing protocol as described in Section 1.3 but use a different kind of dependency graph to determine the recovery lines. In their *local system graph*, when a message sent from checkpoint interval (j, y) is processed in (i, x), an edge is drawn from $c_{j,y+1}$ to $c_{i,x+1}$ as shown in Fig. 4.1(b), in contrast to the corresponding edge $(c_{j,y}, c_{i,x+1})$ in our checkpoint graph. Such a dependency definition can be viewed as extending Johnson and Zwaenepoel's *interval dependency* for state intervals to checkpoint intervals in a non-PWD scenario, i.e., (i, x) depends on (j, y). A significant difference, though, is that such an extension results in possibly cyclic directed graphs. An alternative interpretation can be called the *rollback dependency*, i.e., the rollback of $c_{j,y+1}$ will cause the rollback of $c_{i,x+1}$.

The local system graph corresponding to the checkpoint graph in Fig. 1.4(b) is shown in Fig. 4.1(c). A virtual checkpoint is added at the end of the graph for every process to represent the current state. To determine the global recovery line, all of the virtual checkpoints are initially marked to simulate the situation in which all of the processes are rolled back. All of the checkpoints reachable by these initially marked virtual checkpoints are then searched and marked, and the latest unmarked checkpoint of each process forms

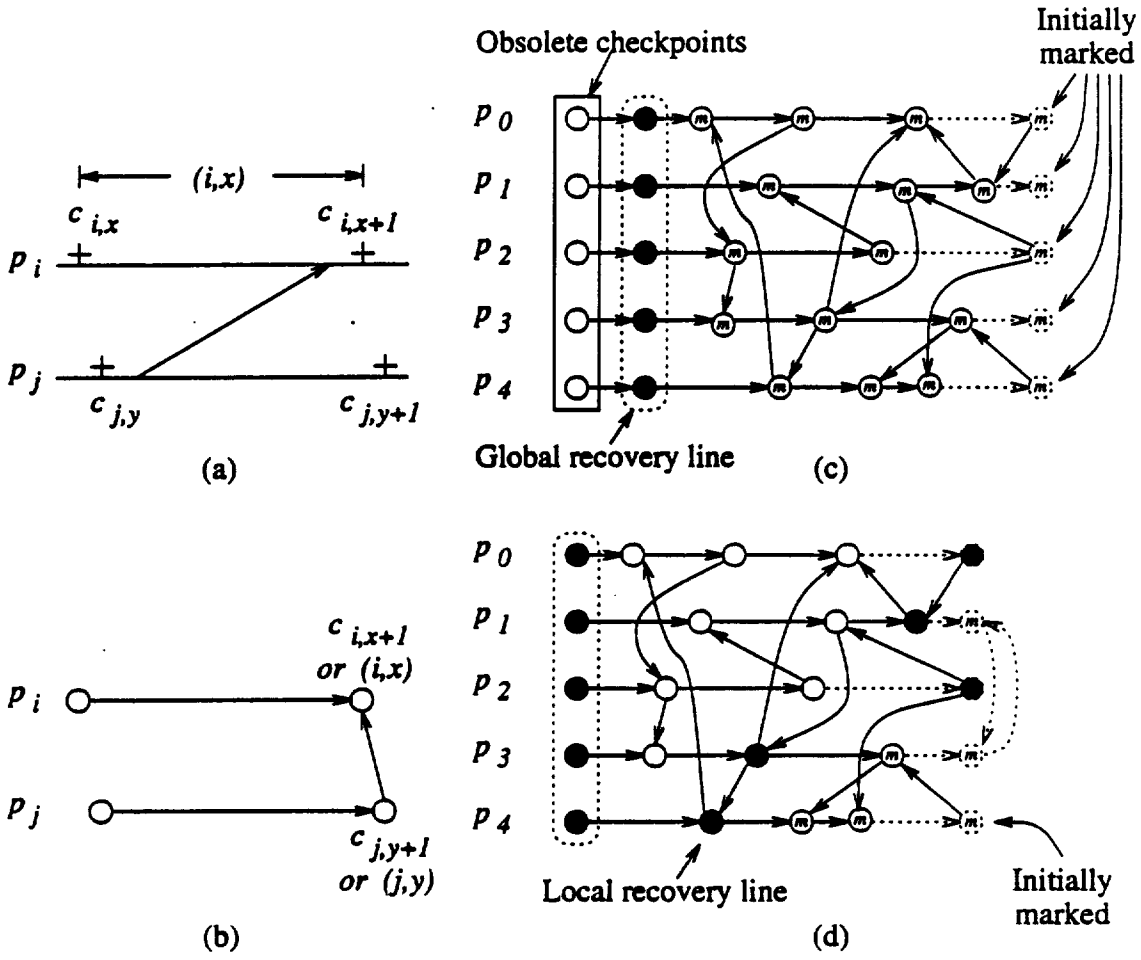


Figure 4.1: Rollback dependency and local system graphs. For the message in (a), (b) illustrates the rollback dependency edge. Local system graphs (c) and (d) are for determining the global and local recovery lines, respectively.

the global recovery line. The local system graph corresponding to the extended checkpoint graph in Fig. 1.4(c) is shown in Fig. 4.1(d). Only the virtual checkpoint belonging to the failed process p_4 is initially marked. The local recovery line again consists of the latest unmarked checkpoint of each process.

4.3 Non-fail-stop Failures and Software Error Recovery

Much of the literature on checkpointing and rollback recovery is based on the assumption of fail-stop hardware failures. As described in Section 1.1, the only cause for rollback propagation under such an assumption is the potential nondeterminism. In practice, hardware failures can be non-fail-stop due to nontrivial error detection latencies; hence, the possibly corrupted messages constitute another source of rollback propagation. One way to build a on-fail-stop recovery protocol on top of a fail-stop recovery protocol is to exclude the potentially corrupted checkpoints of the failed processes from the checkpoint graphs. The rollback propagation algorithm (Fig. 1.5) can then guarantee that all of the possibly corrupted messages and the potentially contaminated checkpoints of the receiving processes do not affect the computation of the correct recovery line. If the maximum error detection latency, possibly different for various types of errors, is known in advance, we can simply exclude the checkpoints belonging to the maximum latency range [61]; otherwise, multiple retries can be performed by discarding more checkpoints when a previous retry fails. For applications in which the *output commit* is an important issue, only the checkpoints and message logs beyond the last output commit can be excluded [26].

Recently, checkpointing and recovery techniques have also been applied to the increasingly important area of software error recovery [42, 62–70]. Unlike the recovery block approach [2] and N-version programming [71] which both use *different programs* to execute on the same set of data, the *on-line retry* approach based on checkpointing

and rollback [65, 66] uses the same program to operate on a *different but consistent set of data* [72, 73] obtained through the inherent nondeterminism, and has been shown to be effective in bypassing software errors to improve system availability [67].

We have proposed a *progressive retry* technique [42] based on the log-based approach for software error recovery. It is based on the observation that, in many long-life software systems, software errors can be recovered by "localized" retries without affecting the other parts of the systems. Therefore, the scope of rollback should be controlled by progressively discarding more and more message log information as a previous retry fails.

We will refer to Fig. 4.2 for the following discussion. First, the failed process p_2 is restarted from a previous checkpoint and replays the logged messages in their original order to reconstruct the process state before the failure. This Step-1 retry will succeed if the error was caused by some transient problems such as concurrency conflicts that may simply disappear after the rollback. When Step-1 retry still leads to an error, the failed process starts a second attempt by reordering the messages; for example, p_2 in Fig. 4.2(b) can reorder M_a and M_b. (We note that M_c becomes an orphan message with respect to the recovery line and hence cannot be used in the reordering.) This Step-2 retry can be useful when the error was due to some untested boundary conditions [63] and the reordering can bypass that condition. In some cases, the software errors are triggered by messages suffering from unexpected transmission delay in the communication channels (message M_d in Fig. 4.2(c)); Step-3 retry thus forces the sender to resend the messages to obtain a "normal" interleaving of messages. If Step-3 still fails, it implies that the

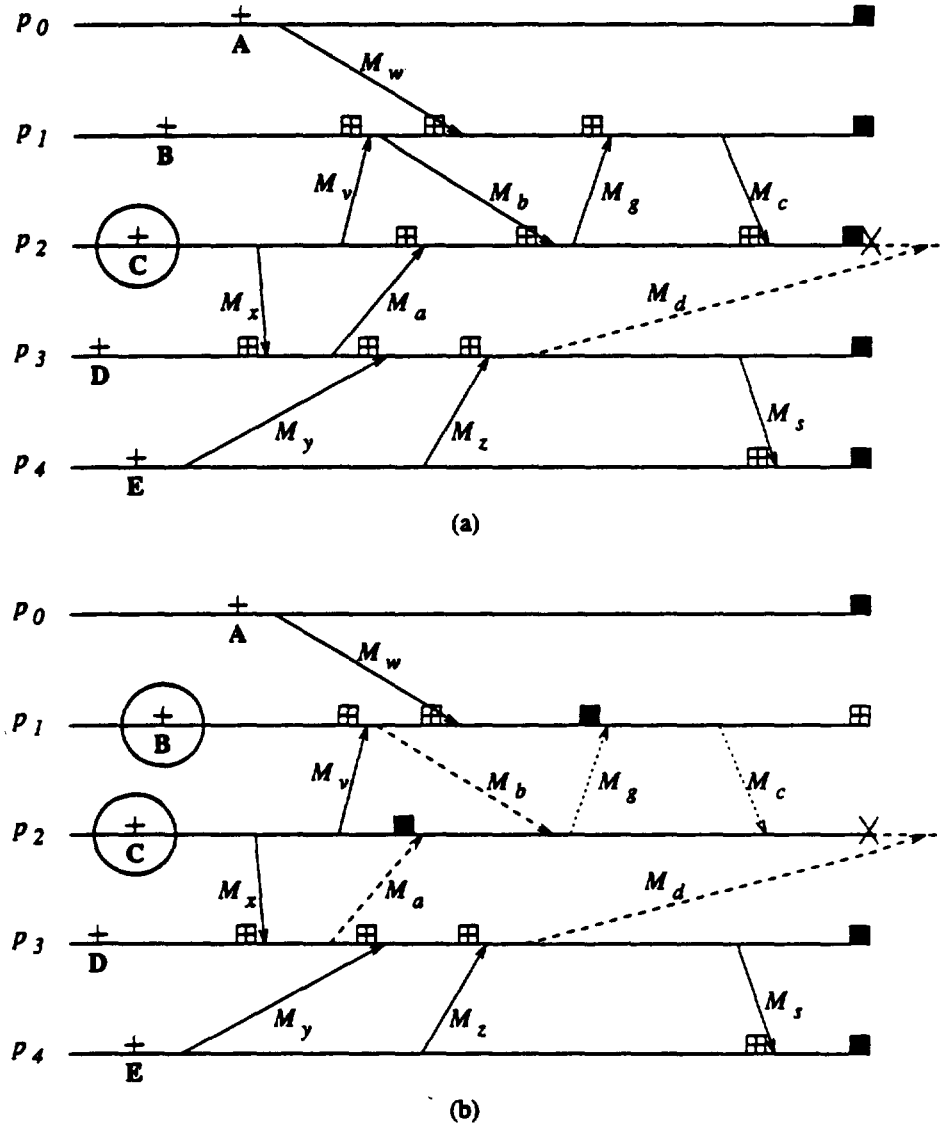
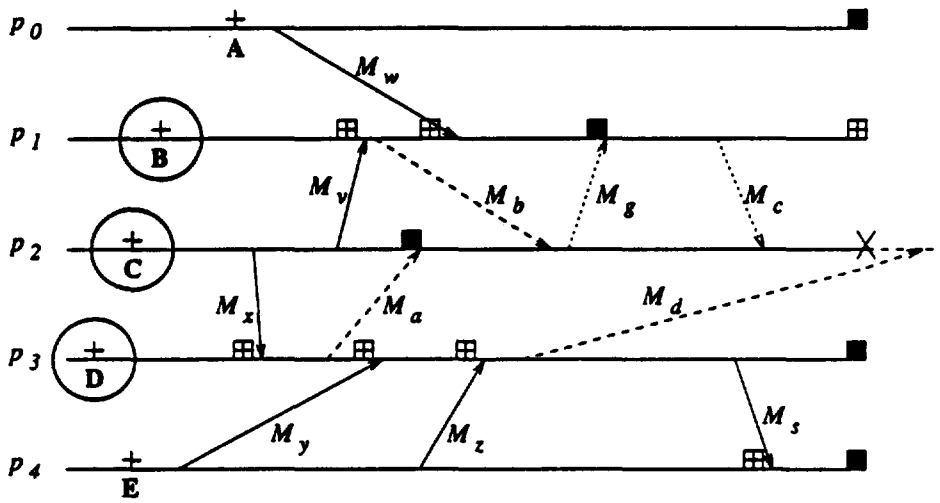
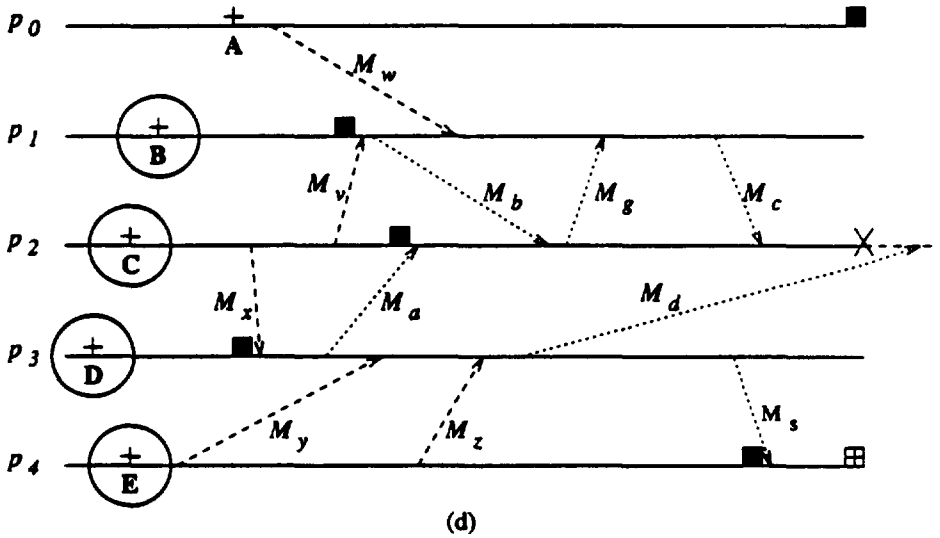


Figure 4.2: Progressive retry. (a) Step 1: message replay (b) Step 2: message reordering (c) Step 3: message resending (d) Step 4: message revocation. (Shaded logical checkpoints indicate the recovery lines; circled physical checkpoints indicate the restarting checkpoints for rolled-back processes; in-transit messages are drawn in dashed lines; orphan messages are drawn in dotted lines.)



(c)



(d)

Figure 4.2: (continued)

above messages may have been corrupted in the first place; Step-4 retry then further rolls back the senders in order to revoke the possibly corrupted messages (M_a and M_b in Fig. 4.2(d)). When all previous retries have failed, Step-5 retry rolls back the entire system to a previous consistent global checkpoint as a final attempt. The progressive retry technique has been used in an AT&T telecommunication billing system and a replicated file system at Bell Laboratories as an economical way of recovering from certain software errors [42, 74].

5. CONCLUSIONS

5.1 Summary

This thesis has derived a necessary and sufficient condition for identifying all garbage checkpoints and message logs in an uncoordinated checkpointing protocol. We proved that there exists a set of N recovery lines such that any checkpoint useful for a possible future recovery must be contained in one of the N recovery lines, and any useful message must be an in-transit message with respect to one of the same N recovery lines. An optimal garbage collection algorithm of time complexity $\mathbf{O}(N|E|)$, where N is the number of processes and $|E|$ is the number of edges in the checkpoint graph, has been presented to identify all nongarbage checkpoints and message logs; the storage space of the remaining checkpoints and message logs can then be reclaimed. In addition, we have demonstrated that the lowest upper bound on the number of nongarbage checkpoints is $N(N+1)/2$, as opposed to the common perception that an uncoordinated checkpointing protocol has to maintain a potentially unbounded number of useful checkpoints.

A unifying framework has also been proposed to integrate the three traditionally separated approaches into one flexible checkpointing and recovery scheme. The framework is based on uncoordinated checkpointing to allow maximum process autonomy and general nondeterministic executions, employs lazy checkpoint coordination to control the coordination frequency and to eliminate the domino effect, and exploits piecewise determinism

whenever possible to further advance the recovery line. It was then demonstrated that the optimal garbage collection algorithm can be applied to such a domino-free recovery protocol to minimize the space overhead.

5.2 Limitations and Future Research

The optimal garbage collection algorithm developed in this thesis is a centralized algorithm based on global dependency information obtained through direct dependency tracking. As demonstrated in Appendix A, in spite of the possible missing *happened before* relations, direct dependency tracking maintains sufficient information for determining consistent global checkpoints. A potential research topic is to study the trade-off between the cost of dependency tracking and the degree of algorithm decentralization. On the one hand, a new dependency tracking mechanism may be devised to record the minimum information sufficient for recovery line computation. Such a scheme would require the collection of global information. On the other hand, transitive dependency tracking [26, 33] or *antecedence graph* tracking [40] may allow decentralized garbage collection based on partial dependency information at the cost of more complicated tracking mechanisms.

A more aggressive approach to reducing space overhead would be to avoid garbage checkpoints in the first place. It is not possible to avoid *taking* a garbage checkpoint because any new checkpoint must be a maximal element in the poset at the time it is taken and hence must be a nongarbage checkpoint according to the algorithm. Such a

checkpoint can become a garbage checkpoint through dependency relations with future checkpoints. Hence, it is possible to insert additional checkpoints based on dependency tracking to avoid *generating* garbage checkpoints. Xu and Netzer [75] have proposed an adaptive checkpointing scheme based on the notion of zigzag paths (as described in Appendix A). An extra checkpoint is inserted whenever a *backward zigzag path* is about to be formed. Since the zigzag paths are, in general, not on-line trackable, causal path tracking is used as an approximation to allow the decision to be made based on local information. For systems allowing message reordering, the message scheduling algorithm as described in Section 3.3 can be combined with the above scheme to reduce the number of extra checkpoints. Alternatively, a checkpoint coordinator which possesses global information may give advice to other processes as to when to take appropriate checkpoints in order not to generate garbage checkpoints. Such an approach can also contribute to advancing the recovery line.

APPENDIX A. MISSING DEPENDENCY IN DIRECT DEPENDENCY TRACKING

Recall that the checkpoint graphs based on direct dependency tracking as described in Section 1.3 record the dependency information, denoted by $<_d^c$, in the following form: $c_{j,y} <_d^c c_{i,x+1}$ if and only if

1. $i = j$ and $x = y$; or

2. $i \neq j$ and there is a message m sent from (j, y) and received in (i, x).

We will denote by $<^c$ the transitive closure of $<_d^c$.

Figure A.1(d) shows the checkpoint graph corresponding to the checkpoint and communication pattern $\mathcal{P}$ in Fig. A.1(a). A close look at Figs. A.1(a) and (d) reveals that the checkpoint graph does not capture the complete *happened before* relation among all checkpoints. For example, while $c_{k,z} < c_{i,x+1}$ is clearly visible in Fig. A.1(a), the corresponding relationship $c_{k,z} <^c c_{i,x+1}$ is absent from Fig. A.1(d). In other words, the poset $R_{\mathcal{P}}^c = (C_{\mathcal{P}}, <^c)$ constructed through direct dependency tracking is not equal to the poset $R_{\mathcal{P}}$ from which we built our model of consistent global checkpoints. This sort of dependency information is lost precisely because of the lack of transitive dependency information propagation in the direct dependency tracking mechanism.

We will demonstrate that, despite the missing dependencies, $R_{\mathcal{P}}^c$ has exactly the same set of maximum-sized antichains as does $R_{\mathcal{P}}$, and therefore that the checkpoint graph

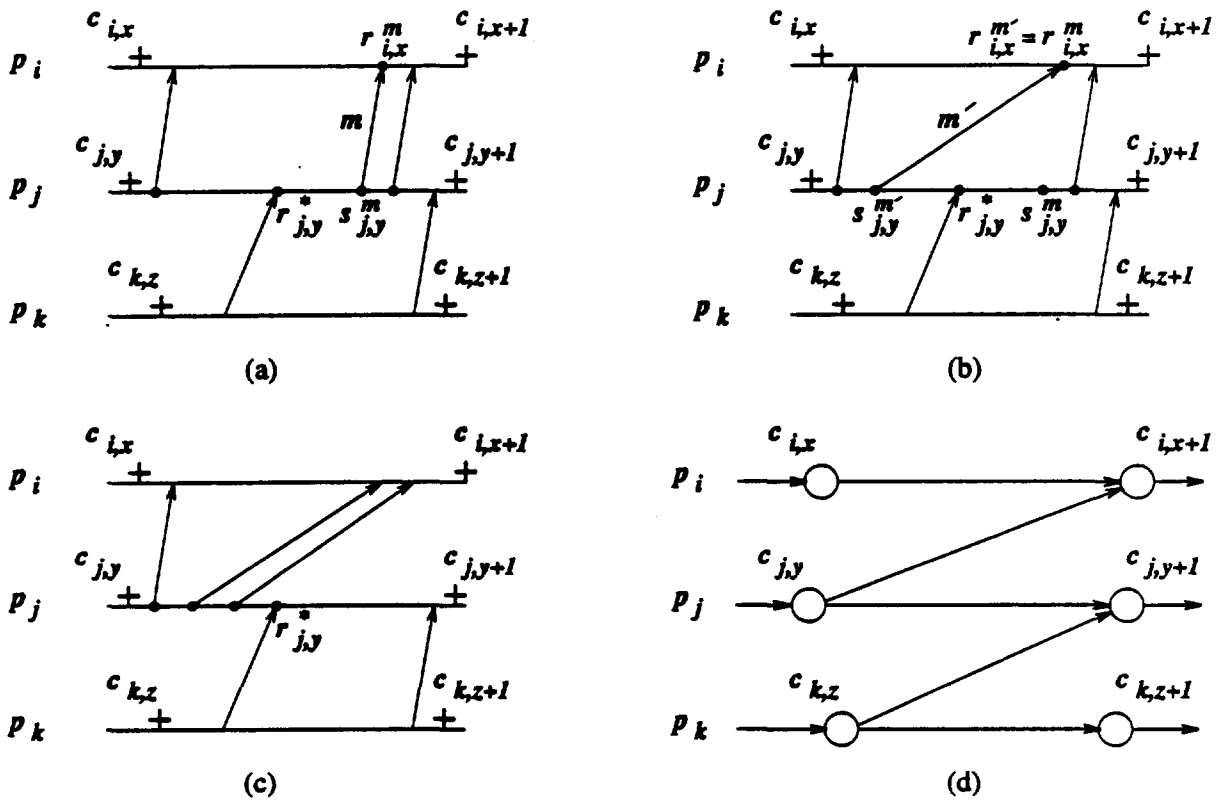


Figure A.1: Three different checkpoint and communication patterns with the same checkpoint graph. The ordering $c_{k,z} < c_{i,x+1}$ implied by (a) and (b) is missing from (d).

suffices to enable the determination of consistent global checkpoints, and of recovery lines in particular. Intuitively, while the *inconsistency* between $c_{k,z}$ and $c_{i,x+1}$ caused by the *happened before* relation $c_{k,z} < c_{i,x+1}$ is missing from Fig. A.1(d), the *global inconsistency* in the sense that $c_{k,z}$ and $c_{i,x+1}$ cannot co-exist in any consistent global checkpoint is implied through the "zigzag" from $c_{k,z}$ to $c_{j,y+1}$ to $c_{j,y}$ to $c_{i,x+1}$.

Xu and Netzer [75, 76] introduce the notion of *zigzag paths* as follows: a zigzag path exists from $c_{j,y}$ to $c_{i,x}$ if and only if there exist messages $m_1, m_2, ..., m_n (n \geq 1)$ such that

1. m_1 is sent by p_j after $c_{j,y}$;

2. if $m_l (1 \leq l < n)$ is processed by p_k in (k, z), then m_{l+1} is sent from (k, z) or a later checkpoint interval (note that m_{l+1} may be sent before or after m_l is processed); and

3. m_n is processed by p_i before $c_{i,x}$.

Figure A.2(a) gives an example of a zigzag path from $c_{j,y}$ to $c_{i,x}$. Figure A.2(b) shows a *causal path* from $c_{j,y}$ to $c_{i,x}$, which is a special case of the zigzag path and results in the *happened before* relation between the two checkpoints. It becomes clear that the notion of zigzag paths is a generalization of Lamport's *happened before* relation to address the "global consistency" issue. In particular, it has been shown that the existence of any zigzag path between two checkpoints excludes the possibility of their belonging to the same consistent global checkpoint, as stated in the following property [75].

PROPERTY 13 *If there exists a zigzag path from $c_{j,y}$ to $c_{i,x}$, then $c_{j,y}$ and $c_{i,x}$ cannot belong to the same consistent global checkpoint.*

It is not hard to see that, if all of the `send` events precede all of the `receive` events in the same checkpoint interval as shown in Fig. A.1(c), then the poset $R^c_{\mathcal{P}}$ corresponding to the transitive closure of the checkpoint graph is exactly the poset $R_{\mathcal{P}}$ corresponding to the checkpoint and communication pattern. Given any checkpoint and communication pattern $\mathcal{P}$, our approach is to transform $\mathcal{P}$ into another pattern $\mathcal{P}^*$ with the above property, without affecting the set of consistent global checkpoints. Denote by $r^*_{j,y}$ the first `receive` event in (j, y) if there is one, or the checkpoint event $c_{j,y+1}$ otherwise.

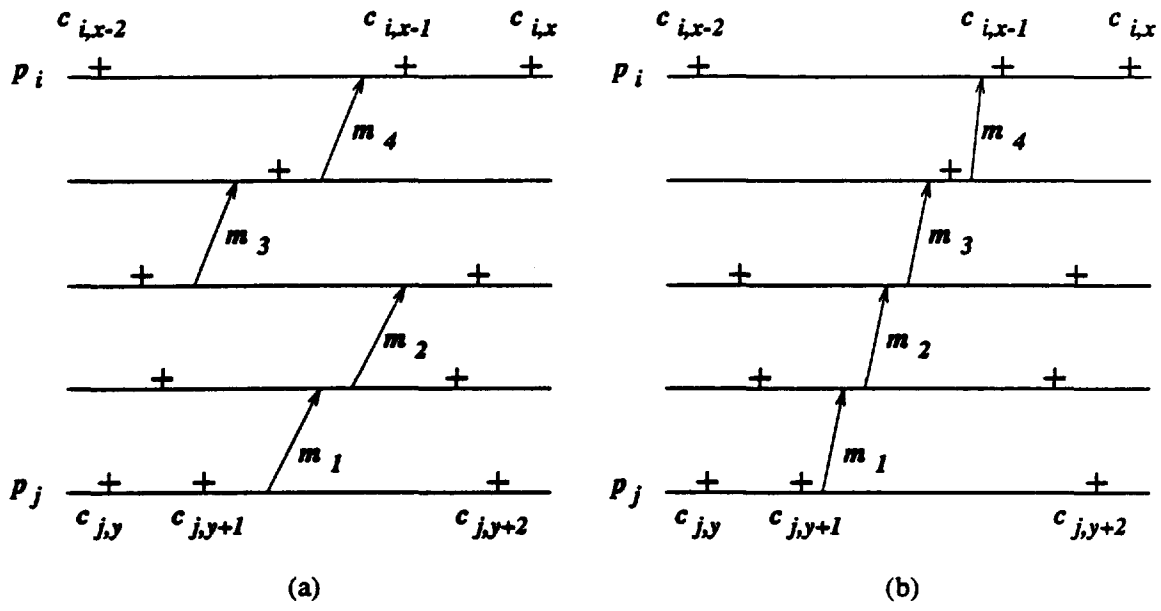


Figure A.2: (a) Zigzag path and (b) causal path.

Let m be a message in $\mathcal{P}$ sent after $r^*_{j,y}$ from (j,y) and processed in (i,x), as shown in Fig. A.1(a). We will denote the **send** and **receive** events for this message by $s^m_{j,y}$ and $r^m_{i,x}$, respectively. The transformation on $\mathcal{P}$ is defined as follows: for each such message m, we first add a message m' with $s^{m'}_{j,y} < r^*_{j,y}$ and $r^{m'}_{i,x} = r^m_{i,x}$ (as shown in fig. A.1(b)), and then remove the message m. We are now prepared to show that the missing dependencies do not affect the determination of consistent global checkpoints.

PROPERTY 14 *For any checkpoint and communication pattern $\mathcal{P}$, $\mathcal{M}(R_{\mathcal{P}}) = \mathcal{M}(R^c_{\mathcal{P}})$. Hence, the poset corresponding to the transitive closure of the checkpoint graph for $\mathcal{P}$ suffices for the determination of consistent global checkpoints of $\mathcal{P}$.*

Proof. It suffices to prove that $\mathcal{M}(R_{\mathcal{P}}) = \mathcal{M}(R_{\mathcal{P}^\bullet})$ because $R_{\mathcal{P}^\bullet} = R^c_{\mathcal{P}^\bullet} = R^c_{\mathcal{P}}$. We consider any global checkpoint M in $R_{\mathcal{P}}$. If $M \in \mathcal{M}(R_{\mathcal{P}})$, then $c_1 \not< c_2$ in $Q_{\mathcal{P}}$ for any

$c_1, c_2 \in M$. Since adding any message m' in the transformation can introduce only the relation $s_{j,y}^{m'} < r_{i,x}^{m'}$ that is already implied in $\mathcal{P}$ through $s_{j,y}^{m'} < s_{j,y}^{m} < r_{i,x}^{m}$, and removing any message m cannot make any originally unordered pair become ordered, we must have $c_1 \not< c_2$ in $Q_{\mathcal{P}^*}$ as well and thus $M \in \mathcal{M}(R_{\mathcal{P}^*})$.

If $M \notin \mathcal{M}(R_{\mathcal{P}})$, there must exist $c_1, c_2 \in M$ such that $c_1 < c_2$ in $Q_{\mathcal{P}}$. Since every message m' in the transformation is sent in the same checkpoint interval as is its corresponding message m, every zigzag path in $\mathcal{P}$ remains a zigzag path in $\mathcal{P}^*$. That $c_1 < c_2$ in $Q_{\mathcal{P}}$ implies there exists a causal path, and hence a zigzag path, from c_1 to c_2. Since that zigzag path must still exist in $\mathcal{P}^*$, we have $M \notin \mathcal{M}(R_{\mathcal{P}^*})$ by Property 13. Therefore, we have shown that $\mathcal{M}(R_{\mathcal{P}}) = \mathcal{M}(R_{\mathcal{P}^*})$ and thus $\mathcal{M}(R_{\mathcal{P}}) = \mathcal{M}(R_{\mathcal{P}}^c)$ as required. □

REFERENCES

[1] D. L. Russel, "State restoration in systems of communicating processes," *IEEE Trans. Software Eng.*, vol. SE-6, pp. 183–194, Mar. 1980.

[2] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Eng.*, vol. SE-1, pp. 220–232, June 1975.

[3] P. A. Lee and T. Anderson, *Fault Tolerance Principles and Practice.* Wien: Springer-Verlag, 1990.

[4] 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.

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

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

[7] K. L. Wu and W. K. Fuchs, "Recoverable distributed shared virtual memory," *IEEE Trans. Comput.*, vol. 39, pp. 460–469, Apr. 1990.

[8] 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 Symp.*, pp. 130–135, 1986.

[9] K. Venkatesh, T. Radhakrishnan, and H. F. Li, "Optimal checkpointing and local recording for domino-free rollback recovery," *Inform. Process. Lett.*, vol. 25, pp. 295–303, July 1987.

[10] Y. Tamir and C. H. Sequin, "Error recovery in multicomputers using global checkpoints," in *Proc. Int. Conf. Parallel Processing*, pp. 32–41, 1984.

[11] K. G. Shin and Y.-H. Lee, "Evaluation of error recovery blocks used for cooperating processes," *IEEE Trans. Software Eng.*, vol. 10, no. 6, pp. 692–700, 1984.

[12] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. Comput. Syst.*, vol. 3, pp. 63–75, Feb. 1985.

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

[14] K. Li, J. F. Naughton, and J. S. Plank, "An efficient checkpointing method for multicomputers with wormhole routing," *Int. J. of Parallel Program.*, vol. 20, pp. 159–180, June 1992.

[15] J. S. Plank, "Efficient checkpointing on MIMD architectures." Ph.D. dissertation, Department of Computer Science, Princeton University, June 1993.

[16] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel, "The performance of consistent checkpointing," in *Proc. IEEE Symp. Reliable Distributed Syst.*, pp. 39–47, Oct. 1992.

[17] 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.

[18] A. Acharya and B. R. Badrinath, "Recording distributed snapshots based on causal order of message delivery," to appear in *Inform. Process. Lett.*, 1992.

[19] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. Software Eng.*, vol. SE-13, pp. 23–31, Jan. 1987.

[20] T. H. Lai and T. H. Yang, "On distributed snapshots," *Inform. Process. Lett.*, vol. 25, pp. 153–158, May 1987.

[21] 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.

[22] Y. M. Wang and W. K. Fuchs, "Lazy checkpoint coordination for bounding rollback propagation," to appear in *Proc. 12th Symp. on Reliable Distributed Syst.*, Oct. 1993.

[23] P. Ramanathan and K. G. Shin, "Checkpointing and rollback recovery in a distributed system using common time base," in *Proc. IEEE Symp. Reliable Distributed Syst.*, pp. 13–21, 1988.

[24] F. Cristian and F. Jahanian, "A timestamp-based checkpointing protocol for long-lived distributed computations," in *Proc. IEEE Symp. Reliable Distributed Syst.*, pp. 12–20, 1991.

[25] Z. Tong, R. Y. Kain, and W. T. Tsai, "Rollback recovery in distributed systems using loosely synchronized clocks," *IEEE Trans. Parallel and Distributed Syst.*, vol. 3, pp. 246–251, Mar. 1992.

[26] R. E. Strom and S. Yemini, "Optimistic recovery in distributed systems," *ACM Trans. Comput. Syst.*, vol. 3, pp. 204–226, Aug. 1985.

[27] 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.

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

[29] A. Borg, W. Blau, W. Graetsch, F. Herrmann, and W. Oberle, "Fault tolerance under UNIX," *ACM Trans. Comput. Syst.*, vol. 7, pp. 1–24, Feb. 1989.

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

[31] 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.

[32] D. B. Johnson and W. Zwaenepoel, "Sender-based message logging," in *Proc. IEEE Fault-Tolerant Computing Symp.*, pp. 14–19, 1987.

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

[34] T. T.-Y. Juang and S. Venkatesan, "Crash recovery with little overhead," in *Proc. IEEE Int. Conf. Distributed Comput. Syst.*, pp. 454–461, 1991.

[35] S. Venkatesan and T. T.-Y. Juang, "Efficient optimistic crash recovery in distributed systems," submitted, 1992.

[36] D. F. Bacon, "Transparent recovery in distributed systems," *ACM Oper. Syst. Review*, pp. 91–94, Apr. 1991.

[37] A. Lowry, J. R. Russell, and A. P. Goldberg, "Optimistic failure recovery for very large networks," in *Proc. IEEE Symp. Reliable Distributed Syst.*, pp. 66–75, 1991.

[38] D. F. Bacon, "File system measurements and their application to the design of efficient operation logging algorithms," *Proc. IEEE Symp. Reliable Distributed Syst.*, pp. 21–30, 1991.

[39] A. P. Goldberg, A. Gopal, K. Li, R. E. Strom, and D. F. Bacon, "Transparent recovery of Mach applications," in *First USENIX Mach Workshop*, Oct. 1990.

[40] E. N. Elnozahy and W. Zwaenepoel, "Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit," *IEEE Trans. Comput.*, vol. 41, pp. 526–531, May 1992.

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

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

[43] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems," *ACM Trans. Comput. Syst.*, vol. 1, pp. 222–238, Aug. 1983.

[44] 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. Submitted to *Inform. Process. Lett.*

[45] K. P. Bogart, *Introductory Combinatorics.* Marshfield, MA: Pitman Publishing Inc., 1983.

[46] I. Anderson, *Combinatorics of Finite Sets.* Oxford: Clarendon Press, 1987.

[47] L. Lamport, "Time, clocks and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, pp. 558–565, July 1978.

[48] V. Hadzilacos, "An algorithm for minimizing roll back cost," in *Proc. ACM Symp. on Principles of Database Systems*, pp. 93–97, 1982.

[49] 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.

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

[51] P. Keleher, A. L. Cox, and W. Zwaenepoel, "Lazy release consistency for software distributed shared memory," in *Proc. Int. Symp. Comput. Architecture*, pp. 13–21, 1992.

[52] A. Gafni, "Rollback mechanisms for optimistic distributed simulation systems," in *Proc. SCS Multiconference on Distributed Simulation*, pp. 61–67, July 1988.

[53] D. B. Johnson and W. Zwaenepoel, "Transparent optimistic rollback recovery," *ACM Oper. Syst. Review*, pp. 99–102, Apr. 1991.

[54] M. Ahamad and L. Lin, "Using checkpoints to localize the effects of faults in distributed systems," in *Proc. IEEE Symp. Reliable Distributed Syst.*, pp. 2–11, 1989.

[55] L. V. Kalé, "The Chare Kernel parallel programming language and system," in *Proc. Int. Conf. Parallel Processing*, pp. II–17–II–25, 1990.

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

[57] K. L. Wu, W. K. Fuchs, and J. H. Patel, "Error recovery in shared memory multiprocessors using private caches," *IEEE Trans. Parallel and Distributed Syst.*, vol. 1, pp. 231–240, Apr. 1990.

[58] R. E. Ahmed, R. C. Frazier, and P. N. Marinos, "Cache-aided rollback error recovery (carer) algorithms for shared-memory multiprocessor systems," in *Proc. IEEE Fault-Tolerant Computing Symp.*, pp. 82–88, 1990.

[59] B. Janssens and W. K. Fuchs, "Experimental evaluation of multiprocessor cache-based error recovery," in *Proc. Int. Conf. Parallel Processing*, pp. I–505–I–508, 1991.

[60] J. Long and W. K. Fuchs, "An evolutionary approach to coordinated checkpointing." submitted to *IEEE Trans. Parallel and Distributed Syst.*, 1992.

[61] L. M. Silva and J. G. Silva, "Global checkpointing for distributed programs," in *Proc. IEEE Symp. Reliable Distributed Syst.*, pp. 155–162, 1992.

[62] J. Gray, "A census of tandem system availability between 1985 and 1990," *IEEE Trans. Reliab.*, vol. 39, pp. 409–418, Oct. 1990.

[63] M. Sullivan and R. Chillarege, "Software defects and their impact on system availability - A study of field failures in operating systems," in *Proc. IEEE Fault-Tolerant Computing Symp.*, pp. 2–9, 1991.

[64] I. Lee and R. K. Iyer, "Faults, symptoms, and software fault tolerance in the tandem guardian90 operating system," in *Proc. IEEE Fault-Tolerant Computing Symp.*, 1993.

[65] J. Gray and D. P. Siewiorek, "High-availability computer systems," *IEEE Comput. Mag.*, pp. 39–48, Sept. 1991.

[66] J. Gray, "Dependable systems." *Keynote Speech, 11th Symp. on Reliable Distr. Syst.*, Oct. 1992.

[67] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques.* San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[68] M. Baker and M. Sullivan, "The recovery box: Using fast recovery to provide high availability in the UNIX environment," in *Proc. Summer '92 USENIX*, pp. 31–43, June 1992.

[69] D. Jewett, "Integrity S2: A fault-tolerant UNIX platform," in *Proc. IEEE Fault-Tolerant Computing Symp.*, pp. 512–519, 1991.

[70] M. N. Meyers, "The AT&T telephone network outage of January 15, 1990." invited talk at *IEEE Fault-Tolerant Computing Symp.*, 1990.

[71] A. Avizienis, "The N-version approach to fault-tolerant software," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 1491–1501, Dec. 1985.

[72] P. E. Ammann and J. C. Knight, "Data diversity: An approach to software fault-tolerance," in *Proc. IEEE Fault-Tolerant Computing Symp.*, pp. 122–126, 1987.

[73] P. E. Ammann and J. C. Knight, "Data diversity: An approach to software fault-tolerance," *IEEE Trans. Comput.*, vol. 37, pp. 418–425, Apr. 1988.

[74] Y. Huang and C. Kintala, "Software implemented fault tolerance: Technologies and experience," in *Proc. IEEE Fault-Tolerant Computing Symp.*, pp. 2–9, June 1993.

[75] J. Xu and R. H. B. Netzer, "Adaptive independent checkpointing for reducing rollback propagation," submitted to *IEEE Symp. Parallel and Distributed Process.*, 1993.

[76] R. H. B. Netzer and J. Xu, "Necessary and sufficient conditions for consistent global snapshots," submitted to *IEEE Trans. Parallel and Distributed Syst.*, 1993.

VITA

Yi-Min Wang was born in [redacted], on [redacted]. He received the B.S. degree in Electrical Engineering from the National Taiwan University, Taipei, Taiwan, in 1986 and was the top-ranked student in his graduating class. From 1988 to 1993, he held a research assistantship at the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign where he received the M.S. degree in Electrical and Computer Engineering in 1990, the Best Presentation Award in the Center for Reliable and High-Performance Computing in 1992, and the Robert T. Chien Memorial Award for his Ph.D. research in 1993.

Yi-Min Wang is a member of Phi Tau Phi and Phi Kappa Phi Honor Societies. After completing his doctoral dissertation, he will be joining AT&T Bell Laboratories at Murray Hill, New Jersey as a Member of Technical Staff. His research interests include fault-tolerant computing, distributed systems, parallel processing and digital signal processing.