REDUCING SPACE OVERHEAD FOR INDEPENDENT CHECKPOINTING

Y.-M. Wang
P.-Y. Chung
I.-J. Lin
W. K. Fuchs

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

Approved for Public Release. Distribution Unlimited.
Reducing Space Overhead for Independent Checkpointing

Uncoordinated checkpointing allows maximum process autonomy and general nondeterministic execution, but suffers from potential domino effect and the associated space overhead. For the past years, checkpoint space reclaimation had been based on the notion of obsolete checkpoints; as a result, a potentially unbounded number of nonobsolete checkpoints may have to be retained on stable storage. In this paper, we derive a necessary and sufficient condition for identifying all garbage checkpoints. By using the approach of recovery line transformation and decomposition, we develop an optimal checkpoint space reclaimation algorithm and show that the space overhead for uncoordinated checkpointing is in fact bounded by \( SN(N+1)/2S \) checkpoints where \( SN \) is the number of processes.
Checkpoint Space Reclamation for Uncoordinated Checkpointing in Message-Passing Systems

Yi-Min Wang
AT&T Bell Laboratories
Murray Hill, NJ 07974

Pi-Yu Chung, In-Jen Lin and W. Kent Fuchs
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Abstract

Uncoordinated checkpointing allows maximum process autonomy and general nondeterministic execution, but suffers from potential domino effect and the associated space overhead. For the past years, checkpoint space reclamation had been based on the notion of obsolete checkpoints; as a result, a potentially unbounded number of nonobsolete checkpoints may have to be retained on stable storage. In this paper, we derive a necessary and sufficient condition for identifying all garbage checkpoints. By using the approach of recovery line transformation and decomposition, we develop an optimal checkpoint space reclamation algorithm and show that the space overhead for uncoordinated checkpointing is in fact bounded by $N(N+1)/2$ checkpoints where $N$ is the number of processes.

---

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

Checkpointing and rollback recovery is an effective approach to tolerating both hardware and software faults. 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 may result in the domino effect [1–3] which prevents recovery line progression.

Numerous checkpointing and rollback recovery techniques have been proposed in the literature for message-passing systems. Uncoordinated checkpointing [4–6] allows maximum process autonomy and general nondeterministic execution. Each process takes its checkpoints independently and keeps track of the dependencies among checkpoints resulted from message communications. When a failure occurs, the dependency information is used to determine the recovery line to which the system should roll back. The major disadvantages of uncoordinated checkpointing have been the potential domino effect and the larger space overhead for maintaining multiple checkpoints of each process.

This paper addresses the second disadvantage by developing an optimal checkpoint space reclamation algorithm to minimize the space overhead. Several techniques have been proposed to address the first issue, i.e., to guarantee recovery line progression. Coordinated checkpointing [7, 8] eliminates the domino effect by sacrificing a certain degree of process autonomy. Extra coordination messages are required to enforce the consistency between the checkpoints belonging to the same checkpointing session. In one experiment [9], it has been shown that the overhead of coordination is reasonably small for a set of benchmark programs, and the overall performance degradation can be made reasonably low if optimization techniques primarily involving changes to the operating systems can be employed. However, for many practical applications, modifying the operating system is not considered a feasible solution. Without being able to use the optimization techniques, application-level checkpointing requires process autonomy in taking checkpoints in order to exploit...
application-dependent information and to checkpoint at the "right time", e.g., when the process state is minimal. Uncoordinated checkpointing is appropriate for such applications. Recently, a lazy checkpoint coordination technique [10] has been developed to eliminate the potential domino effect while still allowing uncoordinated checkpointing. Essentially, the sender's checkpointing progress is piggybacked on each message and the receiver can decide, based on that information, whether it needs to take an extra checkpoint for the purpose of coordination before processing a message, according to a predetermined coordination frequency. Thus, lazy checkpoint coordination and the checkpoint reclamation algorithm developed in this paper together address both disadvantages of uncoordinated checkpointing.

Another approach to eliminating the domino effect is to use the log-based recovery which exploits the piecewise deterministic execution model [11–18]. Under the above model, each process execution is viewed as a number of deterministic state intervals bounded by nondeterministic events. It has been shown that [19] by considering each nondeterministic event log as a logical checkpoint [20] taken at the end of the ensuing state interval, the same dependency model and hence the checkpoint space reclamation algorithm developed in this paper, with little modification, can still be applied.

Traditionally, checkpoint space reclamation 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 checkpoint space reclamation 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.

The outline of the paper is as follows. Section 2 describes the checkpointing and recovery protocol and a model of consistent global checkpoints; Section 3 derives a necessary and sufficient condition for identifying all nongarbage checkpoints and presents the optimal checkpoint space reclamation algorithm; the lowest upper bound on the number of nongarbage checkpoints is derived in Section 4 and experimental evaluation is described in Section 5.

2 Checkpointing and Rollback Recovery

2.1 System Model and Recovery Protocol

The system considered in this paper consists of a number of concurrent processes for which all process communication is through message passing. Processes are assumed to run on fail-stop processors [21]. All processes running on the same recovery unit [13] 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 $x$th checkpoint ($x \geq 0$) of process $p_i$ ($0 \leq i \leq N - 1$). Figure 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 unsend 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, checkpoints $c_{i,x}$ and $c_{j,y}$ in Fig. 1(b) can be considered consistent if $p_i$ can retrieve

\(^1\)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.
the in-transit message $m'$ from a synchronous\(^2\) message log \([11, 12]\) or through a reliable end-to-end transmission protocol \([8]\).

\[
\begin{align*}
&\text{Checkpoint interval } (i,x) \\
p_i &\xrightarrow{m} c_{i,x} \\
p_j &\xrightarrow{} c_{j,y+1}
\end{align*}
\]

(a)

\[
\begin{align*}
&\text{Checkpoint interval } (i,x) \\
p_i &\xrightarrow{c_{i,x}} \\
p_j &\xrightarrow{} c_{j,y+1}
\end{align*}
\]

(b)

Figure 1: Checkpoint consistency. (a) orphan message $m$ with respect to inconsistent checkpoints $c_{i,x+1}$ and $c_{j,y}$; (b) in-transit message $m'$ with respect to consistent checkpoints $c_{i,x}$ and $c_{j,y}$.

During normal execution, each process takes its local checkpoints periodically without coordinating with any other processes. Let $(i, x)$ denote the $x$th checkpoint interval of process $p_i$ between consecutive checkpoints $c_{i,x}$ and $c_{i,x+1}$, as shown in Fig. 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 \([4, 22]\) 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 that are no longer useful for any future recovery. First, $p_i$ broadcasts a request message to collect the direct dependency information from all the 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 2(b) shows the checkpoint graph corresponding to the checkpoint and communication pattern in (a). The rollback propagation algorithm listed in Fig. 3 is executed on the checkpoint graph to determine the global recovery line,\(^3\) which is then broadcast in a recovery line message. All checkpoints before the global recovery line are obsolete, and their space can therefore be reclaimed. Note that upon receiving the request for checkpoint dependency information from the initiator, each process simply supplies the information and then continues with normal execution. The information

---

\(^2\)Extension of our work to an asynchronous logging protocol is considered elsewhere \([6]\).

\(^3\)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.
regarding the garbage checkpoints determined by the initiator is returned later for checkpoint space reclamation.

Figure 2: Checkpointing and rollback recovery. (a) Example checkpoint and communication pattern; (b) checkpoint graph and extended checkpoint graph when $p_0$ initiates a rollback.

/* CP stands for 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 3: The rollback propagation algorithm.
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 2(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 2(d) shows the checkpoint graph immediately after the recovery.

2.2 A Model of Consistent Global Checkpoints

In a message-passing system, event $e_1$ directly happened before event $e_2$ [8] if

- $e_1$ and $e_2$ are events in the same process and $e_1$ occurs immediately before $e_2$; or
- $e_1$ is the sending of a message $m$ and $e_2$ is the receiving of $m$.

The transitive closure of the direct happened before relation is the happened before relation [23], denoted by $\prec$. The set of events with the happened before relation forms a partially ordered set, or poset [23]. For our purpose, we consider only the induced subposet [24] $R = (C, \prec)$, where $C$ is the set of all checkpoints.

Suppose there are $N$ processes in the system. A global checkpoint is defined as a set of $N$ local checkpoints, one from each process. Based on the earlier description of consistency, a consistent global checkpoint is a global checkpoint of which no two constituent checkpoints are ordered through the happened before relation. For the purpose of recovery, we are interested in finding the latest available consistent global checkpoint, referred to as the recovery line, which minimizes the total rollback distance.

Our approach is based on the maximum-sized antichain model for consistent global checkpoints [22]. Given a poset $P = (S, \prec)$, an antichain is a subset $A$ of $S$ such that $x \not\prec y$ for any $x, y \in A$. Intuitively, a consistent global checkpoint corresponds to an antichain of the poset $R = (C, \prec)$. Since the initial checkpoints of all processes must form an antichain of size $N$ and no antichain can contain two checkpoints from the same process, the largest size of any antichain in $R$ is exactly $N$.

The following lemma summarizes the main results described by Wang et al. [22]
LEMMA 1 Given the poset \( R = (C, <) \) of checkpoints generated by the happened before relation and \( M, M_1, M_2 \subseteq C \), let \( \mathcal{M}(R) \) denote the set of maximum-sized antichains of \( R \).

(a) \( M \) is a consistent global checkpoint if and only if \( M \in \mathcal{M}(R) \).

(b) Let \( M[i] \) denote the constituent checkpoint of \( M \) which is a checkpoint of \( p_i \) and, for any \( M_1, M_2 \in \mathcal{M}(R) \), define \( M_1 \preceq M_2 \) if \( M_1[i] \leq M_2[i] \) for all \( 0 \leq i \leq N - 1 \). Then, the poset \( (\mathcal{M}(R), \preceq) \) forms a lattice.

(c) The recovery line is the unique maximal maximum-sized antichain, denoted by \( M^*(R) \), on the lattice \( (\mathcal{M}(R), \preceq) \).

In this paper, we will use the notation \( \mathcal{M}(G) \) to represent the set of maximum-sized antichains of the poset corresponding to the transitive closure of the checkpoint graph \( G \). The notation \( M^*(G) \) is similarly defined.

### 3 Optimal Checkpoint Space Reclamation

#### 3.1 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. 4. 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.

---

\(^4\)It has been pointed out that the poset \( R' \) corresponding to the transitive closure of the checkpoint graph based on direct dependency tracking is not exactly the same as \( R \). However, \( R' \) and \( R \) possess the same set of maximum-sized antichains [22].
Figure 4: 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 5 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\footnote{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. 5 for $c_{0,2}$ or simply invalidates $c_{0,2}$.} and is therefore a garbage checkpoint. Checkpoints $c_{1,1}$ and $c_{0,1}$ are garbage by similar arguments.

Figure 5: Example of nonobsolete garbage checkpoints.

Figure 5 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 4, 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 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.

3.2 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.

3.2.1 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 $G_s(G)$ 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 paper to ensure that the unchanged part, which forms an antichain in $G_1$, remains an antichain in $G_2$ after the transformation.

**Lemma 2** Given a checkpoint graph $G = (V, E)$ and its potential supergraph $G' = (V', E') \in G_s(G)$, 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 $u \not\prec v$ for any $u, v \in A$. Rule 1.b guarantees that $u \not\prec v$ remains true in $G'$ because there cannot 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 paper. 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 6. 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 $G$.

**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

$$M_1 = M^*(G') \cap V \quad \text{and} \quad M_2 = M^*(G') \setminus V$$

as shown in Fig. 7. A corresponding partition of the new-nodes of $G$ is given as $B = B_1 \cup B_2$ such that

$$B_1 = \{n_i : M^*(G')[i] \in M_1\} \quad \text{and} \quad B_2 = \{n_i : M^*(G')[i] \in M_2\}.$$ 

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 $M^*(G')[i] \not\leq_l l_j$ for any $M^*(G')[i] \in M_1$ and $l_j \in L_2$. Now consider $\hat{G} - B_1$. We have $M^*(G')[i] \not\leq_{n_j}$ for any $n_j \in B_2$ because each $n_j$ has only a single incoming edge from $l_j$. Clearly, any new-node $n_j \not\leq M^*(G')[i]$. Lemma 2 further guarantees that $M_1(\subseteq V)$ remains an antichain in $G$ and also in $\hat{G} - B_1$. Hence, we have $M_1 \cup B_2 \in \mathcal{M}(\hat{G} - B_1)$. 

11
Figure 7: Recovery line transformation within an operational session.

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 $M'_1 \subseteq V$, $M_1 \leq M'_1$ and $M_1 \neq M'_1$ as shown in Fig. 7. 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 $u \not\leq M^*(G')[j]$. We also have $M^*(G')[j] \not\leq u$ by Rule 1.b. Therefore, $M'_1 \cup M_2$ forms an 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 $G$. 

12
3.2.2 Transformation across a recovery session

Existing vertices on a checkpoint graph, for example, \( c_{3,3} \) in Fig. 2(c), can be deleted due to rollback recovery. Let \( G_E \) denote the extended checkpoint graph as defined in Section 2, \( G = (V, E) \) denote the subgraph of \( G_E \) without the virtual checkpoints, and \( G^+ = (V^+, E^+) \) denote the checkpoint graph immediately after recovery. Figures 8(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, \( M^*(G_B) \) is the local recovery line. Let \( M^*(G_E) = M_r \cup M_v \) as shown in Fig. 8(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

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

(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 \( l_i, n_i, B, \tilde{G}, T_1 \) and \( T_2 \) for \( G \), we define \( l_i^- \), \( n_i^- \), \( \tilde{G}^- \), \( B^- \), \( T_1^- \) and \( T_2^- \) for \( G^- \). 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 3** 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) \( M^*(G) \preceq M^*(G') \);
Figure 8: Recovery line transformation across a recovery session.
(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 2. In particular, $M^*(G) \in M(G')$ which leads to (b). If $M \subseteq V$ and $M = M^*(G')$, then $M^*(G) \subseteq M$ 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 $G$.

**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

$$M_1 = M^*(G^-_W) \cap V^-,$$  
$$M_2 = \{n^-_i \in M^*(G^-_W) : M^*(G_E)[i] \in M_v\}$$  
$$M_3 = \{n^-_i \in M^*(G^-_W) : M^*(G_E)[i] \in M_r\}$$  

as shown in Fig. 8(f). The two sets of new-nodes $B$ and $B^-$ are partitioned as follows:

$$B = B_1 \cup B_2 \text{ where } B_1 = \{n_i : M^*(G^-_W)[i] \in M_1\} \text{ and } B_2 = \{n_i : M^*(G^-_W)[i] \not\in M_1\}$$

$$B^- = B^-_1 \cup B^-_2 \text{ where } B^-_1 = \{n^-_i : M^*(G^-_W)[i] \in M_1\} \text{ and } B^-_2 = \{n^-_i : M^*(G^-_W)[i] \not\in M_1\}.$$  

Our goal is to show that $M_1 \cup M_2 \cup M_4 = M^*(\hat{G} - (T_1 \cup B_1))$.

where $M_4 = \{M^*(G_E)[i] : n^-_i \in M_3\}$. Then, for any $v \in V^-$ and $v \in M^*(G^-_W)$, we have $v \in M_1 \subseteq M^*(\hat{G} - W)$ where $W = T_1 \cup B_1 \subseteq B$.

---

\(^6\) $T_1 \cup T_2$ (Eq. (1)) is another partition of $B$ corresponding to $M^*(G_E) = M_r \cup M_v$. 

15
First, it is not hard to see that \( W^- \subseteq \overline{B_1} \) and so \( M^\ast(\hat{G}^- - \overline{B_1}) = M^\ast(\overline{G_w}) \) from Lemma 3(c) and the definitions of \( \overline{B_1} \) and \( M_1 \). We now prove Eq. (5) by the following steps: (a) \( M_1 \cup M_2 \cup M_4 \in \mathcal{M}(\hat{G}^- - (T_1^- \cup \overline{B_1})) \); (b) \( M_1 \cup M_2 \cup M_4 \in \mathcal{M}(\hat{G} - (T_1 \cup B_1)) \); (c) \( M_1 \cup M_2 \cup M_4 = M^\ast(\hat{G} - (T_1 \cup B_1)) \). (a) That \( M_1 \cup M_2 \cup M_3 \) forms an antichain in \( \hat{G}^- - \overline{B_1} \) implies, for any \( u \in M_4 \) and \( w \in M_1 \cup M_2 \), that \( w \not< u \). This clearly remains true in \( \hat{G}^- - (T_1^- \cup \overline{B_1}) \). Since Rule 2.a guarantees \( u \not< w \), we have \( M_1 \cup M_2 \cup M_4 \in \mathcal{M}(\hat{G}^- - (T_1^- \cup \overline{B_1})) \). (b) By adding all of the vertices in \( F \) to \( \hat{G}^- - (T_1^- \cup \overline{B_1}) \), we obtain the graph \( \hat{G} - (T_1 \cup B_1) \) as shown in Fig. 8(d). Rule 2.b guarantees that the above process will not make any unordered pair in \( \hat{G}^- - (T_1^- \cup \overline{B_1}) \) 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 \( M_1 \cup M_2 \cup M_4 \neq M^\ast(\hat{G} - (T_1 \cup B_1)) \). Then, we must have

\[
M_1 \cup M_2 \cup M_4 \leq M^\ast(\hat{G} - (T_1 \cup B_1)).
\]

By applying the transformation as defined in the proof of Property 1 to graphs \( G \) and \( G_E \) (Fig. 8(b)), we have

\[
M^\ast(\hat{G} - T_1) = M_r \cup T_2,
\]

as shown in Fig. 8(e). Since \( \hat{G} - T_1 \) is a potential supergraph of \( \hat{G} - (T_1 \cup B_1) \), we have, by Lemma 3(b),

\[
M^\ast(\hat{G} - (T_1 \cup B_1)) \leq M^\ast(\hat{G} - T_1).
\]

Equations (6), (7) and (8) and the fact that \( M_2 \subseteq T_2 \) and \( M_4 \subseteq M_r \) imply \( M_2 \cup M_4 \subseteq M^\ast(\hat{G} - (T_1 \cup B_1)) \), and there exists \( M_1' \) such that \( M_1' = M^\ast(\hat{G} - (T_1 \cup B_1)) \setminus (M_2 \cup M_4) \), \( M_1 \leq M_1' \) and \( M_1 \neq M_1' \) (as shown in Fig. 8(d)). Equations (7) and (8) further guarantee that \( M_1' \) does not intersect \( F \) and so must exist in \( \hat{G}^- - (T_1^- \cup \overline{B_1}) \) and hence \( \hat{G}^- - \overline{B_1} \). 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 \( M_1 \cup M_2 \cup M_3 = M^\ast(\hat{G}^- - \overline{B_1}) \).
3.2.3 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 $v$ 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 $q$th operational session and $G'$ belongs to the $r$th session where $r \geq q$. Let $G_i$ denote the checkpoint graph at the end of the $i$th 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 $(G', G^-)$, $(G_j - W_j, G^-_j)$ 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 $W \subseteq B$. \hfill \Box

Figure 9 gives an example demonstrating the recovery line transformation. Figure 9(a) is the current checkpoint graph $G$ considered for garbage collection. Suppose that Fig. 9(b) is the extended checkpoint graph when $p_3$ initiates a rollback, then Figure 9(c) is the checkpoint graph immediately after the recovery. Fig. 9(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. 9(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 $B$.

### 3.3 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 $X \land Y$ 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 4** Given a poset $P$, $M \in \mathcal{M}(P)$ and $M \leq M_i \in \mathcal{M}(P)$ for $0 \leq i \leq k - 1$ for any finite $k$, define $\bigwedge_{0 \leq i \leq k-1} M_i = (((M_0 \land M_1) \land M_2) \ldots \land M_{k-1}$, then

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

**Proof.** See Appendix.

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

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

**Proof.** Without loss of generality, let $W = \{n_0, n_1, \ldots, n_{k-1}\}$ where $1 \leq k \leq N$. Since $\hat{G} - n_j \in \mathcal{G}_e(\hat{G} - W)$, $M^*(\hat{G} - W) \leq M^*(\hat{G} - n_j)$ for all $0 \leq j \leq k - 1$ by Lemma 3(b).

Now consider the graph $\hat{G}$. From Lemma 3(a), we have $M^*(\hat{G} - W) \in \mathcal{M}(\hat{G})$ and $M^*(\hat{G} - n_j) \in \mathcal{M}(\hat{G})$. Therefore,$M^*(\hat{G} - W)$ is an element of the basis of recovery lines, so $M^*(\hat{G} - W) \leq \min(\bigcup_{n_i \in W} M^*(\hat{G} - n_i)$. Hence, $M^*(\hat{G} - W) = \min \left( \bigcup_{n_i \in W} M^*(\hat{G} - n_i) \right)$.
Figure 9: Example recovery line transformation.
$M(\hat{G})$ 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 4, we have

$$M^*(\hat{G} - W) \leq \bigwedge_{0 \leq j \leq k - 1} M_n^*(\hat{G} - n_j) = M^*_n \in M(\hat{G}). \quad (10)$$

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 3(a), we have \( M^*_n \in M(\hat{G} - W) \) and hence

$$M_n^* \leq M^*(\hat{G} - W). \quad (11)$$

Combining Eqs. (10) and (11), we have proved that

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

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

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

As an example, we demonstrate the decomposition of \( M^*(G_e) \) in Fig. 9(e) where \( G_e = \hat{G} - \{n_0, n_1, n_3, n_4\} \). From Property 4 and referring to Fig. 10, we have

$$M^*(G_e) = \min(M^*(\hat{G} - n_0) \cup M^*(\hat{G} - n_1) \cup M^*(\hat{G} - n_3) \cup M^*(\hat{G} - n_4))$$

$$= \min(\{A, B, n_2, n_3, n_4, n_0, I, n_1, J, C, D\}) = \{A, B, n_2, C, D\}$$

which is exactly the recovery line shown in Fig. 9(e).

### 3.4 Predictive checkpoint space reclamation algorithm

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

**THEOREM 1** A checkpoint \( v \) in a checkpoint graph \( G \) is nongarbage if and only if \( v \in M^*(\hat{G} - n_i) \) for some \( 0 \leq i \leq N - 1 \).
Figure 10: Example of the PCSR algorithm. Shaded checkpoints in (a)-(e) belong to the recovery lines and the nonshaded checkpoints in (f) are garbage.
Proof. If \( v \in M^*(\hat{G} - n_i) \) for some \( 0 \leq i \leq N - 1 \), then \( v \) is nongarbage because \( \hat{G} - n_i \) is a possible future checkpoint graph. Conversely, if \( v \) is nongarbage, we have by definition \( v \in M^*(G') \) for some future checkpoint graph \( G' \). From Property 3, \( v \in M^*(\hat{G} - W) \) for some \( W \subseteq B \); from Property 4,

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

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

Based on Theorem 1 we now present the Predictive Checkpoint Space Reclamation (PCSR) algorithm for finding the \( N \) recovery lines in Fig. 11. Since the rollback propagation algorithm in Fig. 3 is of time complexity \( O(|E|) \) 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 \( O(N|E|) \).

```c
/* \( N_g(G) \) denotes the set of nongarbage checkpoints of \( G \) */
/* \( N \) is the number of processes */
/* \( \hat{G} \) and \( n_i \) are as defined in Fig. 6 */

for each \( 0 \leq i \leq N - 1 \) {
    apply the rollback propagation algorithm in Fig. 3 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 11: The Predictive Checkpoint Space Reclamation algorithm.

An example illustrating the execution of the PCSR algorithm on the checkpoint graph \( G \) in Fig. 6 is shown in Fig. 10. 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. 10(f) can be discarded.
4 Lowest Upper Bound on Number of Nongarbage Checkpoints

As mentioned in the Introduction, traditional approach to checkpoint space reclamation by discarding only obsolete checkpoints has lead to the common perception that the space overhead for uncoordinated checkpointing is potentially unbounded. Theorem 1 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 3(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 3(a), $M^*(\hat{G} - n_i) \in \mathcal{M}(\hat{G} - n_j)$. We then have $M^*(\hat{G} - n_i) \leq M^*(\hat{G} - n_j)$. Similarly, $M^*(\hat{G} - n_j)[i] \neq n_i$ leads to $M^*(\hat{G} - n_j) \leq M^*(\hat{G} - n_i)$. Therefore, $M^*(\hat{G} - n_i) = M^*(\hat{G} - n_j)$. □

We are now prepared to prove the second major result of this paper.

**THEOREM 2** 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 1, 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 $M^*(\hat{G} - n_j)[i] = M^*(\hat{G} - n_i)[i]$. 

23
Since $M^*(\hat{G} - n_j)[j]$ and $M^*(\hat{G} - n_i)[i]$ are already in $N_\delta(G)$, this case does not increase the size of $N_\delta(G)$.

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

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

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. 12 to achieve this upper bound. Figure 12 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.

![Figure 12: $G_N^*$: The checkpoint graph with $N(N + 1)/2$ nongarbage checkpoints.](image)

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 [8] 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.
5 Trace-Driven Simulation 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 [25]. 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 1.

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

Figures 13-16 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 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.
Figure 13: Nonobsolete and nongarbage checkpoints for Cell Placement.

Figure 14: Nonobsolete and nongarbage checkpoints for Channel Router.
Figure 15: Nonobsolete and nongarbage checkpoints for Knight Tour.

Figure 16: Nonobsolete and nongarbage checkpoints for N-Queen.
6 Summary

We have derived a necessary and sufficient condition for identifying all garbage checkpoints 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. An optimal checkpoint space reclamation algorithm of time complexity $O(N|E|)$, 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; the storage space for the remaining checkpoints 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. Communication trace-driven simulation for four parallel programs demonstrated that the algorithm can be effective in significantly reducing the number of retained checkpoints.

7 Appendix

Proof of Lemma 4. Both parts will be proved by induction on $k$ and based on the following theorem from Anderson's book [26]: 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

$$M_1 \land M_2 = \min(M_1 \cup M_2). \quad (12)$$

(a) $\mathcal{M}(P)$ is a lattice and therefore $M_0 \land M_1 \in \mathcal{M}(P)$. Also, $M \leq M_0 \land M_1$ because $M \leq M_0$, $M \leq M_1$ and $M_0 \land M_1$ 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.,

$$M \leq \bigwedge_{0 \leq i \leq n-2} M_i \in \mathcal{M}(P). \quad (13)$$

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

$$\bigwedge_{0 \leq i \leq n-1} M_i = (\bigwedge_{0 \leq i \leq n-2} M_i) \land M_{n-1} \in \mathcal{M}(P).$$
Equation (13) and $M \leq M_{n-1}$ imply that

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

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

(b) The case $k = 2$ follows directly from Eq. (12). Assume that it is true for $k = n - 1$, i.e.,

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

(14)

Applying part (a), Eqs. (12) and (14), we have

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

Lemma 5 (to be proved next) further gives that

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

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

**LEMMA 5** Given a poset $P = (S, <)$ and $A, B \subseteq S$, $\min(A \cup B) = \min(\min(A) \cup B)$.

**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 $z \notin \min(\min(X) \cup Y)$.

□
Acknowledgement

The authors wish to express their sincere thanks to Douglas West (Illinois) and Weiping Shi (North Texas) for their discussions; to Michael Loui (Illinois), Andy Lowry (IBM), Kang Shin (Michigan), Robert Strom (IBM), and the anonymous referees for their valuable comments; to Balkrishna Ramkumar (Iowa) and Junsheng Long (North Carolina) for their help with the experimental results; to L. V. Kalé (Illinois) for access to the Chare Kernel and to Prith Banerjee (Illinois) for use of his hypercube programs.

References


