SUBMITTED TO: IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS

CHECKPOINT SPACE RECLAMATION FOR INDEPENDENT CHECKPOINTING IN MESSAGE-PASSING SYSTEMS

Yi-Min Wang, Pi-Yu Chung, In-Jen Lin and W. Kent Fuchs
University of Illinois at Urbana-Champaign

ABSTRACT

The main disadvantages of independent checkpointing in message-passing systems are the possible domino effect and the associated storage space overhead for maintaining multiple checkpoints. In most previous research on checkpointing and recovery, it has been assumed that only the checkpoints older than the global recovery line can be discarded. In this paper, we generalize the notion of a recovery line to that of a potential recovery line. Only the checkpoints belonging to at least one of the potential recovery lines can not be discarded. By using the model of maximum-sized antichains on a partially ordered set, an efficient algorithm is developed for finding all non-discardable checkpoints and an upper bound on the number of non-discardable checkpoints is derived. Communication trace-driven simulation for several parallel programs is used to show the benefits of the proposed algorithm for real applications.

PREFERRED ADDRESS FOR CORRESPONDENCE

Yi-Min Wang

Center for Reliable and High-Performance Computing

Coordinated Science Laboratory

1101 W. Springfield Ave.

University of Illinois

Urbana, IL 61801

E-mail: ymwang@crhc.uiuc.edu

Phone: (217) 244-7161

FAX: (217) 244-5686

Statement A per telecom
Dr. Clifford Lau
ONR/Code 1114
Arlington, VA 22217-5000 NWW 6/23/92

INDEX TERMS

checkpoint space reclamation

fault tolerance

independent checkpointing

message-passing systems

recovery lines

rollback recovery

FOOTNOTES

Affiliation of Authors:

Yi-Min Wang and W. Kent Fuchs are with Center for Reliable and High-Performance Computing, University of Illinois, Urbana, IL 61801.

Pi-Yu Chung is with Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

In-Jen Lin is with the Department of Mathematics, University of Illinois, Urbana, IL 61801.

Acknowledgement of Financial Support:

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).

Numbered Footnotes:

1. Our goal is similar to the idea of discarding useless recovery points at earlier possible time as described in the papers by Kim et. al [1, 2].

2. Extension of our work to systems with optimistic logging protocols is considered elsewhere [3].

3. the comparability through the "happens before" relation.

4. The subscript "r" stands for real checkpoints and "v" for virtual checkpoints.

CAPTIONS FOR FIGURES AND TABLES

Figure 1: Checkpoint consistency. (a) message received but not yet sent; (b) message sent but not yet received.

Figure 2: Example checkpoint graph. (a) the checkpoint and communication pattern; (b) the corresponding checkpoint graph.

Figure 3: The rollback propagation algorithm.

Figure 4: (a) The extended checkpoint graph when p_0 initiates the recovery; (b) CP_{22} and all virtual checkpoints are deleted from the checkpoint graph after the recovery.

Figure 5: Construction of the potential supergraph $\hat{G}$ by adding n_i's to G.

Figure 6: Transforming the potential recovery line by replacing M_2 with B_2.

Figure 7: Transforming the potential recovery line by replacing M_3 with M_4.

Figure 8: The Predictive Checkpoint Space Reclamation Algorithm.

Figure 9: Execution of the PCSR algorithm (a) $\hat{G} - n_0$ (b) $\hat{G} - n_1$ (c) $\hat{G} - n_2$ (d) $\hat{G} - n_3$. (Shaded checkpoints belong to the recovery lines and checkpoints marked "X" are discardable.)

Figure 10: G^*_V: The checkpoint graph with $N(N+1)/2$ non-discardable checkpoints.

Figure 11: Non-obsolete checkpoints and non-discardable checkpoints for the Cell placement program.

Figure 12: Non-obsolete checkpoints and non-discardable checkpoints for the Channel router program.

Figure 13: Non-obsolete checkpoints and non-discardable checkpoints for the Knight tour program.

Figure 14: Non-obsolete checkpoints and non-discardable checkpoints for the N-queen program.

Table 1: Execution and checkpoint parameters of the programs.

I INTRODUCTION

Numerous checkpointing and rollback recovery techniques have been proposed in the literature for message-passing systems. They can be classified into two basic categories. *Coordinated checkpointing* schemes synchronize computation with checkpointing by coordinating processors during a checkpointing session in order to maintain a consistent set of checkpoints [4–6]. Each processor only keeps the most recent successful checkpoint and rollback propagation is avoided at the cost of potentially significant performance degradation during normal execution. *Independent checkpointing* schemes replace the above synchronization by dependency tracking and possibly message logging [7–10] in order to preserve process autonomy. Possible rollback propagation in case of a fault is handled by searching for a consistent system state based on the dependency information. Lower run-time overhead during normal execution is achieved by maintaining multiple checkpoints and allowing slower recovery.

This paper considers the independent checkpointing schemes for possibly nondeterministic execution. Most research on this subject has concentrated on algorithms for finding the latest consistent set of checkpoints, i.e., the *recovery line*, during rollback recovery. The same algorithms can be applied to the set of existing checkpoints during normal execution to find the global recovery line. All the checkpoints older than the global recovery line then become obsolete checkpoints and can therefore be discarded. When the domino effect [11] occurs, a potentially large number of non-obsolete checkpoints have to be kept on the stable storage and result in large space overhead.

Our approach is based on the observation that many non-obsolete checkpoints can also be discarded because they will never become members of any future recovery line[1]. The notion of a recovery line is generalized to that of a potential recovery line. A checkpoint is non-discardable if and only if it belongs to at least one of the potential recovery lines. By modeling a recovery line as the maximal maximum-sized antichain on a partially ordered

[1] Our goal is similar to the idea of discarding useless recovery points at the earlier possible time described in the papers by Kim et. al [1, 2].

set, an efficient algorithm is developed for finding the set of non-discardable checkpoints. An upper bound on the size of this set given the number of processors is also derived to show that even when domino effect persists during program execution, the space overhead for maintaining multiple checkpoints will not grow without limit.

The outline of the paper is as follows. Section II describes the system model and the checkpointing and recovery protocol; Section III gives our mathematical model of recovery lines; Section IV presents the necessary and sufficient conditions for a checkpoint to be non-discardable; the checkpoint space reclamation algorithm is developed in Section V; the maximum number of non-discardable checkpoints is derived in Section VI and the experimental evaluation is described in Section VII.

II SYSTEM MODEL AND CHECKPOINT CONSISTENCY

A. Checkpointing and Rollback 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 [12] and each processor is considered as an individual recovery unit [8]. Since studies [13] have shown that the support for nondeterministic processes is important for practical applications, we do not assume deterministic execution. Consequently, if the sender of a message is rolled back, the corresponding message will be invalid during reexecution, which means that the receiver also has to be rolled back. We do not address the problem of concurrent rollbacks due to multiple failures [9].

During normal execution, the state of each processor is occasionally saved as a *checkpoint* on stable storage and can be reloaded for rollback recovery in case of a detected error. Let CP_{ik} denote the kth checkpoint of processor p_i with $k \geq 0$ and $0 \leq i \leq N-1$, where N is the number of processors. A *checkpoint interval* is defined to be the time between two

consecutive checkpoints on the same processor and the interval between CP_{ik} and $CP_{i(k+1)}$ is called the kth checkpoint interval. Each message is tagged with the current checkpoint interval number and the processor number of the sender. Each processor takes its checkpoint independently, i.e., without synchronizing with any other processors. Each checkpoint includes *communication information* (or *input information* [9]) containing pairs of the processor number and checkpoint interval number, (j, m), if at least one message from the mth checkpoint interval of processor p_j has been received during the previous checkpoint interval.

A centralized *checkpoint space reclamation algorithm* can be invoked by any processor periodically to reduce the space overhead by removing discardable checkpoints. First, the communication information of all the existing checkpoints is collected to construct the *checkpoint graph*. The *rollback propagation algorithm* (described later) is executed on the checkpoint graph to determine the *global recovery line* [7]. All the checkpoints taken before the global recovery line then become *obsolete* and their space can therefore be reclaimed.

When processor p_i detects an error, it starts a two-phase centralized recovery procedure [9]. First, a *rollback-initiating* message is sent to every other processor to request the up-to-date communication information. Each surviving processor takes a *virtual checkpoint* upon receiving the *rollback-initiating* message so that the communication information during the most recent checkpoint interval is also collected. After receiving the responses, p_i constructs the *extended checkpoint graph* [7] and executes the rollback propagation algorithm to determine the *local recovery line*. A *rollback-request* message containing the local recovery line is then sent to each processor and requests the involved processors to rollback and restart.

B. Checkpoint Consistency

There are two situations concerning the consistency between two checkpoints. In Fig. 1(a), if processors p_i and p_j restart from the checkpoints CP_{ik} and CP_{jm} respectively, the message m is recorded as "received but not yet sent". In a general model without the assumption of deterministic execution, message m becomes an *orphan message* [14]. CP_{ik} and CP_{jm} are thus inconsistent.

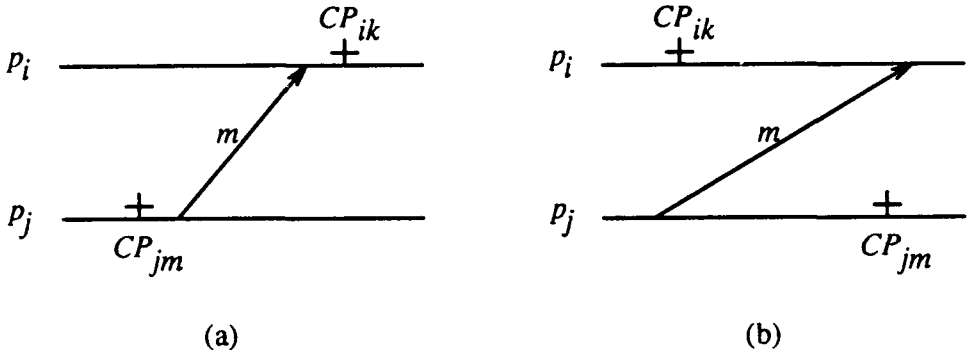

Figure 1: Checkpoint consistency. (a) message received but not yet sent; (b) message sent but not yet received.

Fig. 1(b) illustrates the second situation. The message m is recorded as "sent but not yet received" according to the system state containing CP_{ik} and CP_{jm}. By defining the *state of the channels* to be the set of messages sent but not yet received, it has been shown [5, 15] that checkpoints like CP_{ik} and CP_{jm} can be considered consistent if the corresponding state of the channels is also recorded. In Koo and Toueg's paper [6], such a state was assumed to be recorded at the sender side in the form of lost messages and the set of messages was guaranteed to be re-delivered reliably by some end-to-end transmission protocol. Another way of recording the channel state is through message logging. Pessimistic logging protocols [16–18] can ensure such a state is properly recorded at the receiving end[2]. As a result, we consider the situation in Fig. 1(b) as consistent.

III RECOVERY LINES

A. Partially Ordered Sets and Checkpoint Graphs

In a message-passing system, an event a *happens before* event b [19] if and only if

1. a and b are events on the same processor, and a occurs before b; or

[2]Extension of our work to systems with optimistic logging protocols is considered elsewhere [3].

2. a is the sending of a message by one processor and b is the receiving of the same message by another processor; or

3. a happens before c and c happens before b.

The set of events with the "happens before" relation forms a *partially ordered set*, or *poset* [19]. When dealing with the problem of finding a consistent set of checkpoints, we only consider the *induced subposet* [20] $P = (C, <)$, where C is the set of all checkpoints and "$<$" is the "happens before" relation.

A *checkpoint graph (CPG)*, of which the transitive closure is the poset P, is a directed acyclic graph constructed as follows [7]. Each vertex on the checkpoint graph represents a checkpoint. A directed edge exists from vertex CP_{ik} to vertex CP_{jm} if $j = i$ and $m = k + 1$, or $j \neq i$ and there exists a message sent from the kth checkpoint interval of processor p_i and received by processor p_j at the $(m-1)$th checkpoint interval. Fig. 2 gives an example of CPG with its corresponding communication pattern.

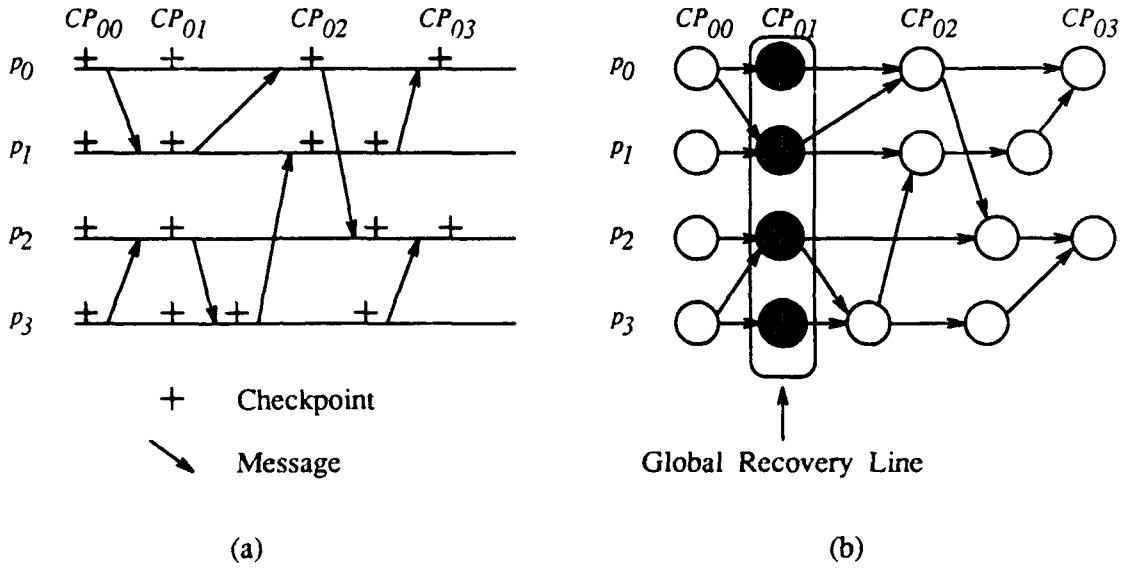

Figure 2: Example checkpoint graph. (a) the checkpoint and communication pattern; (b) the corresponding checkpoint graph.

Most of the ideas in this paper will be illustrated by the CPG instead of the more abstract poset. An element a in a poset is *maximal (minimal)* if there does not exist any element b such that $a < b$ ($b < a$); correspondingly, a vertex in a CPG will be referred to as maximal (minimal) if it has no outgoing (incoming) edge. Also, the following terminology will be used interchangeably: $a < b$, a is "smaller than" b, b is "greater than" a, a can "strictly reach" b and b is "strictly reachable from" a.

B. Maximum-Sized Antichains and Recovery Lines

A partial ordering of a set S is *linear* if for every two elements a and b in S, either $a < b$ or $b < a$ [20]. In a poset, a subset whose elements are linearly ordered is called a *chain* and a set of elements, no two of which are comparable, is called an *antichain*. In particular, a set of any number of maximal (minimal) elements clearly forms an antichain. The antichains with the largest number of elements are called the *maximum-sized antichains* or *M-chains* for short. Let $\mathcal{A}(Q)$ denote the set of antichains on a poset Q and, for $A, B \in \mathcal{A}(Q)$, define $A \preceq B$ if and only if for all $a \in A$ there exists $b \in B$ such that $a \leq b$. [21]. Also let $\mathcal{M}(Q)$ denote the set of maximum-sized antichains. We then have the following properties [21, 22].

LEMMA 1

(a) $(\mathcal{A}(Q), \preceq)$ forms a poset;

(b) $(\mathcal{A}(Q), \preceq)$ is a lattice and its subposet $(\mathcal{M}(Q), \preceq)$ is a sublattice;

(c) For $M_1, M_2 \in \mathcal{M}(Q)$, the **join** *(least upper bound) $M_1 \vee M_2 = max(M_1 \cup M_2)$ and the* **meet** *(greatest lower bound) $M_1 \wedge M_2 = min(M_1 \cup M_2)$, where $max(S)$ denote the set of all maximal elements in S and $min(S)$ is similarly defined.*

Since $(\mathcal{M}(Q), \preceq)$ is a finite lattice, there must exist a unique maximum member $M^*(Q)$ [21], called the *maximal maximum-sized antichain* [23] or *MM-chain*, such that $M \preceq M^*(Q)$ for every $M \in \mathcal{M}(Q)$.

LEMMA 2 *For any $M \in \mathcal{M}(Q)$, there must not exist any $a \in M^*(Q)$ such that $a < b$ for some $b \in M$.*

Proof. Suppose there exist such $a \in M^*(Q)$ and $b \in M$. $M \preceq M^*(Q)$ implies there exists $c \in M^*(Q)$ such that $b \leq c$. Together with $a < b$, this leads to $a < c$, contradicting the fact that $M^*(Q)$ is an antichain. □

In this paper, we define a *global checkpoint* as a set of N checkpoints, one from each processor. Based on the description of consistency in the previous section, a *consistent global checkpoint* is a set of checkpoints, one from each processor and no two of which are comparable through the "happens before" relation. A *recovery line* refers to the "most recent" consistent global checkpoint. Since one special feature of the poset $P = (C, <)$ is that there always exists a natural *chain decomposition* [21] $\{C_0, C_1, ..., C_{N-1}\}$ where C_i consists of all checkpoints of processor p_i, the size $d(P)$ of the M-chains cannot be greater than N. Furthermore, because the first checkpoint of every processor must be minimal and the set of such checkpoints always forms an antichain of size N, $d(P)$ is in fact equal to N and each M-chain will consist of N elements, one from each C_i. It becomes clear that each M-chain is equivalent to a consistent global checkpoint. Since it is always desirable to rollback to the most recent consistent global checkpoint in order to minimize the recovery cost, Lemma 1 guarantees the existence and uniqueness of such a recovery line, i.e., the MM-chain.

C. Ideals, Filters and the Reachable Sets

Given a poset P, if $\mathcal{I}$ is a set of elements of P with the property

$$a \in \mathcal{I} \text{ and } b \leq a \Longrightarrow b \in \mathcal{I},$$

$\mathcal{I}$ is called an *ideal* or a *down-set* of P. Similarly, a *filter* or an *up-set*, $\mathcal{F}$, of P is a set of elements such that if $a \in \mathcal{F}$ and $a \leq b$, then $b \in \mathcal{F}$.

For an antichain A in P, define

$$\mathcal{I}(A) = \{x \in P : x \leq a \text{ for some } a \in A\}$$

$$\mathcal{F}(A) = \{x \in P : a \leq x \text{ for some } a \in A\}.$$

Then $\mathcal{I}(A)$ is an ideal [21] and $\mathcal{F}(A)$ is a filter.

LEMMA 3 *A and B are antichains, then [21]*
(a) $\mathcal{I}(A) \subseteq \mathcal{I}(B) \Longleftrightarrow A \preceq B$;
(b) $\mathcal{F}(A) \subseteq \mathcal{F}(B) \Longleftrightarrow B \preceq A$.

In terms of the CPG, the set of vertices which can reach any vertex in an antichain A is equal to $\mathcal{I}(A)$ and the set of vertices reachable from any vertex in A is equal to $\mathcal{F}(A)$.

D. The Rollback Propagation Algorithm

The algorithm for finding the recovery line will form the basis of our checkpoint space reclamation algorithm. The problem of finding the MM-chain in a general poset can be transformed into that of finding a maximum matching on a bipartite graph [23]. For the poset $P = (C, <)$ in our problem, a simpler *rollback propagation algorithm*, shown in Fig. 3, has been proposed [7] and applied to the CPG. The complexity of the algorithm is linear in the number of edges because each edge can be removed after it is used to reach some vertex and therefore visited at most once.

```
/* CP stands for checkpoint */
/* Initially, all the CPs are unmarked */
include the latest CP of each processor 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 processor;
    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.

Having introduced the checkpoint graphs and the rollback propagation algorithm, we now give an example illustrating the checkpoint space reclamation algorithm and the recovery protocol. The global recovery line for the checkpoint graph in Fig. 2(b) is indicated by the shaded vertices. The four checkpoints before the global recovery line are obsolete. Suppose the extended checkpoint graph when p_0 initiates the recovery is as shown in Fig. 4(a). The checkpoint graph after recovery (Fig. 4(b)) is then obtained by deleting the virtual checkpoints and all the checkpoints after the local recovery line.

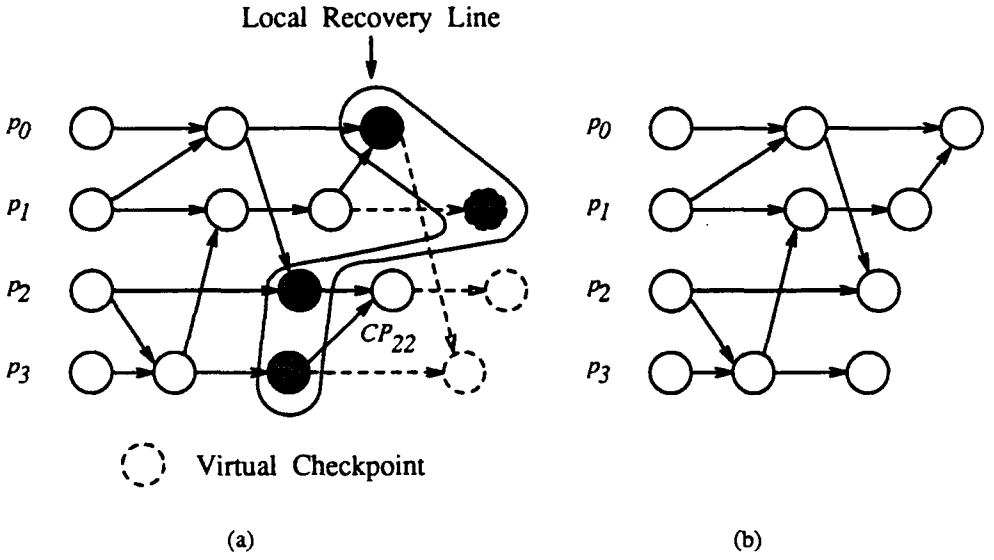

Figure 4: (a) The extended checkpoint graph when p_0 initiates the recovery; (b) CP_{22} and all virtual checkpoints are deleted from the checkpoint graph after the recovery.

IV POTENTIAL RECOVERY LINES

Let $\mathcal{G}_f(G)$ denote the set of all *future graphs* of a checkpoint graph G which can possibly evolve from G during program execution in the future. We define a *potential recovery line* of G as the recovery line of some checkpoint graph G' where $G' \in \mathcal{G}_f(G)$. Since the purpose of keeping checkpoints is for possible future recovery, a checkpoint is *discardable* if and only

if it does not belong to any potential recovery line. Being obsolete is simply a sufficient condition for being discardable but not a necessary condition. We will show that there exist discardable non-obsolete checkpoints.

Although the execution time for a typical program is finite, the number of future graphs of any given checkpoint graph is enormous because the communication pattern and the error occurrence are in general unpredictable. By characterizing the possible evolution of a checkpoint graph, we are able to reduce the almost infinite number of situations to a manageable number of finite cases for the problem of identifying the minimum number of non-discardable checkpoints.

A. Adjoining New Vertices During Normal Execution

During normal execution, the size of the checkpoint graph increases as new checkpoints are taken. Because checkpoint graphs represent program dependency, the following rules must be satisfied when adding new vertices. For each new vertex CP_{ik} with $k \geq 1$,

Rule 1: CP_{ik} must have an incoming edge from $CP_{i(k-1)}$;

Rule 2: CP_{ik} can have incoming edges from an arbitrary number of existing vertices. But it can not have any outgoing edge to any existing vertex.

Note that a checkpoint CP_{ik} that "happens before" CP_{jm} may not be collected before CP_{jm} because of the unpredictable message transmission delay during the collection process. However, such a situation can be detected by the communication information. If a vertex CP_{jm} is expecting an incoming edge from a non-existing vertex CP_{ik}, CP_{jm} and its associated incoming edges will be excluded from the existing CPG. By adding each new vertex under this constraint, none of the new vertices can have edges pointing to any existing vertex and, therefore, Rule 2 is enforced. The following important property is ensured by Rule 2.

PROPERTY 1 *Adding a new vertex v and its associated incoming edges to an existing CPG can not change the relation*[3] *between any pair of existing vertices.*

[3]the comparability through the "happens before" relation.

Proof. The relation between any pair of existing vertices will be changed only if one vertex is smaller than v and the other one is greater than v. However, Rule 2 guarantees none of the existing vertices is greater than v. Therefore, the property holds. □

Let $\mathcal{G}_s(G)$ denote the set of all *potential supergraphs* obtainable by adjoining new vertices to a given checkpoint graph G according to the above rules. Lemma 4 gives the relationship between the antichains of G and those of its potential supergraphs.

LEMMA 4 *Given checkpoint graphs* $G = (V, E)$ *and* $G' \in \mathcal{G}_s(G)$,
(a) $\mathcal{A}(G) \subseteq \mathcal{A}(G')$;
(b) $A \in \mathcal{A}(G')$ *and* $A \subseteq V \Longrightarrow A \in \mathcal{A}(G)$;
(c) $\mathcal{M}(G) \subseteq \mathcal{M}(G')$;
(d) $M \in \mathcal{M}(G')$ *and* $M \subseteq V \Longrightarrow M \in \mathcal{M}(G)$;
(e) $M^*(G) \preceq M^*(G')$.

Proof. (a) and (b) follow immediately from Property 1. By Rule 1 and the discussion after Lemma 2, the size of the maximum-sized antichains is always fixed and equal to the number of processors, thus (c) and (d) holds. In particular, $M^*(G) \in \mathcal{M}(G) \subseteq \mathcal{M}(G')$ implies $M^*(G) \preceq M^*(G')$. □

One special potential supergraph of G, $\hat{G}$, will play a very important role throughout this paper and is constructed as follows:

1. adjoin N new vertices $n_0, n_1, \ldots, n_{N-1}$ to G;

2. an edge is added from the last vertex e_i on each chain C_i to n_i as shown in Fig. 5.

Let U denote the set of all such n_i's. We now prove the following theorem which relates the MM-chain of any potential supergraph to the MM-chain of one of the subgraphs of $\hat{G}$.

LEMMA 5 *Given a checkpoint graph* $G = (V, E)$ *and* $v \in V$, *if* $v \in M^*(G')$ *for some* $G' = (V', E') \in \mathcal{G}_s(G)$, *then* $v \in M^*(\hat{G} - W)$ *for some* $W \subseteq U$.

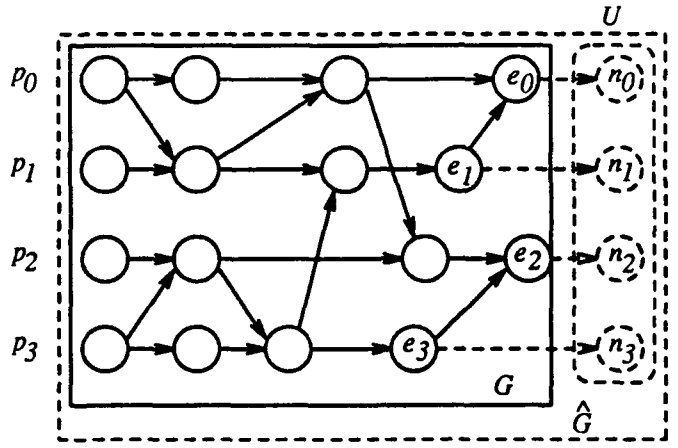

Figure 5: Construction of the potential supergraph $\hat{G}$ by adding n_i's to G.

Proof. If $v \in M^*(G')$ for some $G' \in \mathcal{G}_s(G)$, let $M^*(G') = M_1 \cup M_2$ such that $M_1 = M^*(G') \cap V$ and $M_2 = M^*(G') \setminus M_1$ as shown in Fig. 6(a). Clearly, $v \in M_1$. Define $p(u) = i$ if u represents a checkpoint of processor p_i and partition the set U as $U = B_1 \cup B_2$ where

$$B_1 = \{n_{p(u)} : u \in M_1\}, \quad B_2 = \{n_{p(u)} : u \in M_2\}.$$

We want to show that $M_1 \cup B_2 = M^*(\hat{G} - B_1)$ (Fig. 6(b)).

First we prove $M_1 \cup B_2 \in \mathcal{M}(\hat{G} - B_1)$. Consider the graph G'. For every $u \in M_2$, $e_{p(u)} < u$ by Rule 1. According to Lemma 3, $\mathcal{I}(e_{p(u)}) \subseteq \mathcal{I}(u)$. Since u and all the vertices in M_1 belong to the same antichain, $M_1 \cap \mathcal{I}(u) = \emptyset$. It follows that $M_1 \cap \mathcal{I}(e_{p(u)}) = \emptyset$. Now consider the graph $\hat{G} - B_1$. $M_1 \cap \mathcal{I}(e_{p(u)}) = \emptyset$ still holds because of Property 1. By the construction of $\hat{G}$, $\mathcal{I}(n_{p(u)}) = \mathcal{I}(e_{p(u)}) \cup \{n_{p(u)}\}$ and therefore $M_1 \cap \mathcal{I}(n_{p(u)}) = \emptyset$. Since $n_{p(u)}$ is maximal and so $M_1 \cap \mathcal{F}(n_{p(u)}) = \emptyset$, we have proved that every vertex $n_{p(u)}$ in B_2 is incomparable with every vertex in M_1. Because M_1 is an antichain by itself and all $n_{p(u)}$'s in B_2 are maximal, $M_1 \cup B_2 \in \mathcal{M}(\hat{G} - B_1)$.

Next we prove $M_1 \cup B_2 = M^*(\hat{G} - B_1)$ by contradiction. Because every vertex in B_2 is maximal on the chain it belongs to, $B_2 \subseteq M^*(\hat{G} - B_1)$ by Lemma 2. 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 $M_1 \neq M_1'$. We then have $\mathcal{F}(M_1') \subseteq \mathcal{F}(M_1)$ by Lemma 3. Now consider the graph G'.

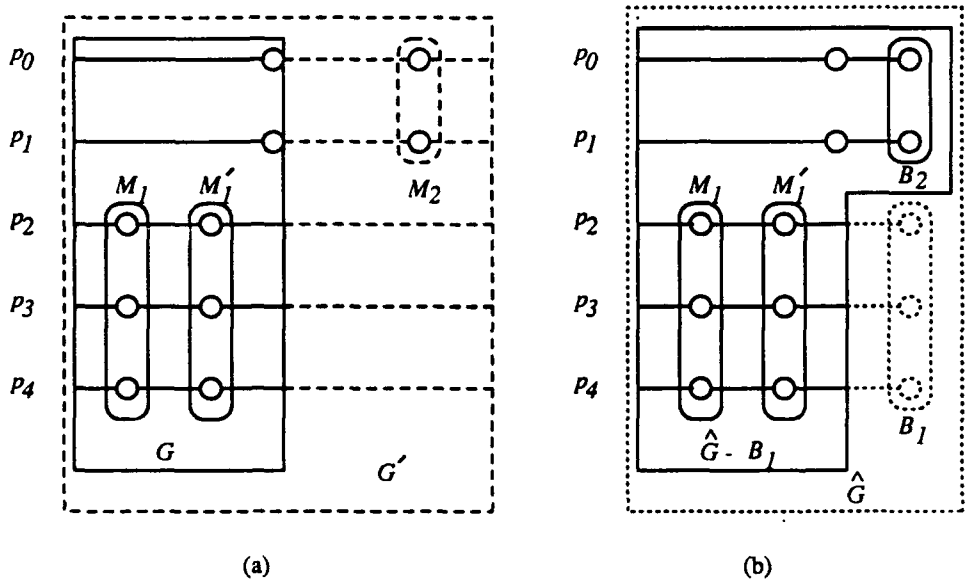

Figure 6: Transforming the potential recovery line by replacing M_2 with B_2.

Recall M_1 and M_2 form an antichain in the graph G', which implies $M_2 \cap \mathcal{F}(M_1) = \emptyset$. Thus $M_2 \cap \mathcal{F}(M_1') = \emptyset$ and, because Rule 2 guarantees $M_2 \cap \mathcal{I}(M_1') = \emptyset$, $M_1' \cup M_2 \in \mathcal{M}(G')$. The fact that $M_1' \cup M_2$ is a greater M-chain than $M_1 \cup M_2$ in G' contradicts $M^*(G') = M_1 \cup M_2$. Hence, $M_1 \cup B_2 = M^*(\hat{G} - B_1)$. It immediately follows that if $v \in M^*(G')$ for some $G' \in \mathcal{G}_s(G)$, $v \in M_1 \subseteq M^*(\hat{G} - B_1)$ for $B_1 \subseteq U$. □

The proof of Lemma 5 shows that although the number of potential supergraphs of a checkpoint graph G is infinite, the recovery lines of these potential supergraphs can only intersect G in a finite number of ways. Furthermore, each of the possible intersections must be part of the recovery line for one of the 2^N graphs $\hat{G} - W$, $W \subseteq U$.

B. Deleting Vertices During Rollback Recovery

Existing vertices on a checkpoint graph, for example CP_{22} in Fig. 4, may be deleted due to rollback recovery. Let G_e denote the extended CPG during recovery, $G = (V, E)$ denote the subgraph of G_e without the virtual checkpoints, and $G^- = (V^-, E^-)$ denote the CPG immediately after recovery. By definition, $M^*(G_e)$ is the local recovery line. Define a *strict*

filter corresponding to an antichain A in a poset P as

$$\mathcal{F}_s(A) = \mathcal{F}(A) \setminus A.$$

According to the recovery protocol, we have $G^- = G - \mathcal{F}_s(M^*(G_e))$. Let $M^*(G_e) = M_r \cup M_v$ where $M_r = M^*(G_e) \cap V$ and $M_v = M^*(G_e) \setminus M_r$[4], and define

$$T_1 = \{n_{p(u)} : u \in M_r\}, \quad T_2 = \{n_{p(u)} : u \in M_v\}.$$

In other words, T_1 contains vertex n_i for each processor p_i which contributes a real checkpoint to the local recovery line. Parallel to the definitions of e_i, n_i, U, $\hat{G}$, T_1 and T_2 for G, we define e_i^-, n_i^-, U^-, $\hat{G}^-$, T_1^- and T_2^- for G^-. Clearly, $T_2^- = T_2$. The following lemma gives several properties of G and G^-.

PROPERTY 2

(a) For any $u \in M_r$, u is maximal in G^-;

(b) $M^*(\hat{G} - T_1) = M_r \cup T_2$;

(c) $G \in \mathcal{G}_s(G^-)$.

Proof. (a) For any $u \in M_r$, u is on the local recovery line. All the vertices "greater than", or "strictly reachable from", u are deleted from the checkpoint graph by the rollback propagation algorithm. and therefore u is maximal in G^- after recovery.

(b) Since $G_e \in \mathcal{G}_s(G)$ and $M^*(G_e) = M_r \cup M_v$, we can show $M_r \cup T_2$ forms the MM-chain of $\hat{G} - T_1$ by following the proof of Lemma 5.

(c) In order to prove $G \in \mathcal{G}_s(G^-)$, we have to show that all the vertices in $\mathcal{F}_s(M^*(G_e))$ and their associated edges can be added back to G^- to reconstruct G without violating Rules 1 and 2. Rule 1 is obviously satisfied. By always adding the "smaller" vertices first, Rule 2 is enforced among the vertices in $\mathcal{F}_s(M^*(G_e))$ during the process. Suppose Rule 2 is violated when $v \in \mathcal{F}_s(M^*(G_e))$ is being added, i.e., there exists $u \in G^-$ such that an edge is drawn from v to u. By the definition of $\mathcal{F}_s(M^*(G_e))$, $v < u$ and $v \in \mathcal{F}_s(M^*(G_e))$ implies

[4]The subscript "r" stands for real checkpoints and "v" for virtual checkpoints.

$u \in \mathcal{F}_s(M^*(G_e))$, contradicting the fact that $G^- \cap \mathcal{F}_s(M^*(G_e)) = \emptyset$. Therefore, Rule 2 is also satisfied and $G \in \mathcal{G}_s(G')$. □

We now prove Lemma 6 which shows a way to transform a recovery line of $\hat{G}^- - W^-$, $W^- \subseteq U^-$, to a recovery line of $\hat{G} - W$, $W \subseteq U$.

LEMMA 6 *For checkpoint graphs $G = (V, E)$ and $G^- = (V^-, E^-)$ as defined at the beginning of this subsection and any $v \in V^-$, if $v \in M^*(\hat{G}^- - W^-)$ for some $W^- \subseteq U^-$, then $v \in M^*(\hat{G} - W)$ for some $W \subseteq U$.*

Proof. Partition $M^*(\hat{G}^- - W^-) = M_1 \cup M_2 \cup M_3$ where $M_1 = M^*(\hat{G}^- - W^-) \cap V^-$, $M_2 = M^*(\hat{G}^- - W^-) \setminus M_1 \setminus T_1^-$ and $M_3 = M^*(\hat{G}^- - W^-) \setminus M_1 \setminus M_2$. Clearly, $v \in M_1$. Fig. 7 illustrates the above notation. Also define

$$M_4 = \{e^-_{p(u)} : n^-_{p(u)} \in M_3\}$$

$$B = \{n_{p(u)} : u \in M_1.\}$$

We want to prove that $M_1 \cup M_2 \cup M_4 = M^*(\hat{G} - (T_1 \cup B))$.

First we show $M_1 \cup M_2 \cup M_4 \in \mathcal{M}(\hat{G} - (T_1 \cup B))$. By the definition of M_4, $M_4 \preceq M_3$ and thus $\mathcal{I}(M_4) \subseteq \mathcal{I}(M_3)$. Since $M_1 \cup M_2 \cup M_3$ forms an antichain, $(M_1 \cup M_2) \cap \mathcal{I}(M_3) = \emptyset$. So $(M_1 \cup M_2) \cap \mathcal{I}(M_4) = \emptyset$. Now consider the graph $\hat{G}^- - (T_1^- \cup B)$. $\mathcal{F}(M_4) = \emptyset$ because vertices in M_4 are all maximal according to Property 2(a). Therefore, $M_1 \cup M_2 \cup M_4 \in \mathcal{M}(\hat{G}^- - (T_1^- \cup B))$. It is not hard to see that $G \in \mathcal{G}_s(G^-)$ (Property 2(c)) implies $\hat{G} - (T_1 \cup B) \in \mathcal{G}_s(\hat{G}^- - (T_1^- \cup B))$. By Lemma 4(c), $M_1 \cup M_2 \cup M_4 \in \mathcal{M}(\hat{G} - (T_1 \cup B))$.

Now suppose $M_1 \cup M_2 \cup M_4 \neq M^*(\hat{G} - (T_1 \cup B))$. Since $\hat{G} - T_1 \in \mathcal{G}_s(\hat{G} - (T_1 \cup B))$, $M^*(\hat{G} - (T_1 \cup B)) \preceq M^*(\hat{G} - T_1)$ by Lemma 4(e). The fact that $M_2 \cup M_4 \subseteq T_2 \cup M_r = M^*(\hat{G} - T_1)$ (Property 2(b)) and $M_1 \cup M_2 \cup M_4 \in \mathcal{M}(\hat{G} - (T_1 \cup B))$ implies that $M_2 \cup M_4 \subseteq M^*(\hat{G} - (T_1 \cup B))$. Therefore, there must exist $M_1' = M^*(\hat{G} - (T_1 \cup B)) \setminus (M_2 \cup M_4)$ such that $M_1 \preceq M_1'$ and $M_1 \neq M_1'$. In fact, $M_1' \subseteq V^-$ because $M_1' \preceq M^*(\hat{G} - (T_1 \cup B)) \preceq M^*(\hat{G} - T_1)$. Now consider the graph $\hat{G}^- - W^-$. $\mathcal{F}(M_1') \subseteq \mathcal{F}(M_1)$ and $(M_2 \cup M_3) \cap \mathcal{F}(M_1) = \emptyset$ lead

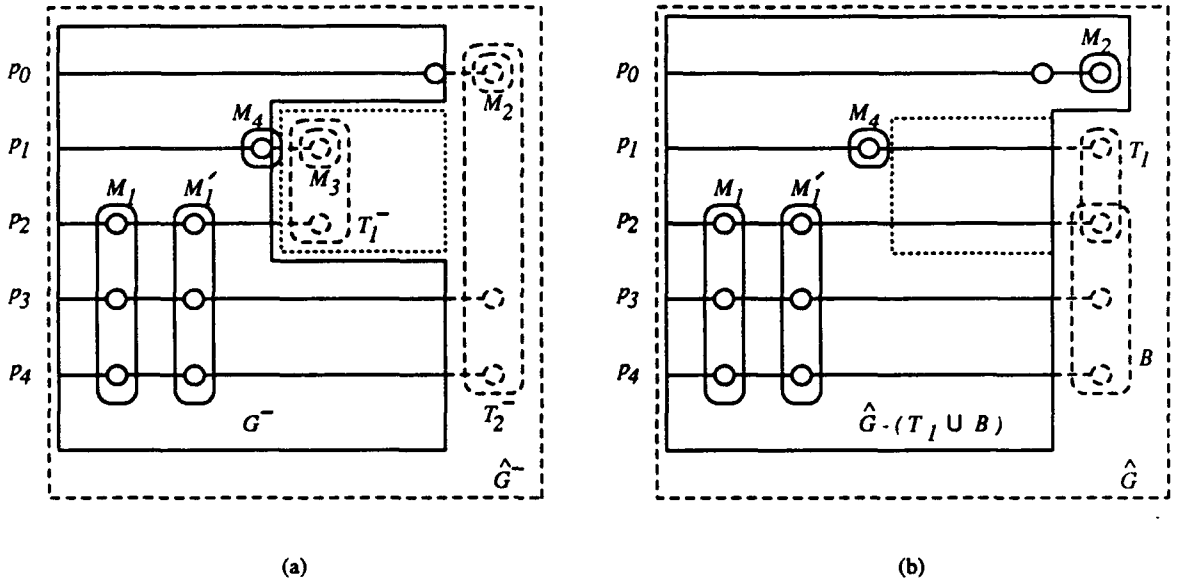

Figure 7: Transforming the potential recovery line by replacing M_3 with M_4.

to $(M_2 \cup M_3) \cap \mathcal{F}(M_1') = \emptyset$. Since $M_2 \cup M_3 \subseteq U^-$ and $M_1' \subseteq V^-$, $(M_2 \cup M_3) \cap \mathcal{I}(M_1') = \emptyset$. Therefore, $M_1' \cup M_2 \cup M_3 \in \mathcal{M}(\hat{G}^- - W^-)$, $M_1 \cup M_2 \cup M_3 \preceq M_1' \cup M_2 \cup M_3$ and $M_1 \cup M_2 \cup M_3 \neq M_1' \cup M_2 \cup M_3$, contradicting the fact that $M^*(\hat{G}^- - W^-) = M_1 \cup M_2 \cup M_3$. Therefore, $M_1 \cup M_2 \cup M_4 = M^*(\hat{G} - (T_1 \cup B))$. Finally, we have $v \in M_1 \subseteq M^*(\hat{G} - W)$ where $W = T_1 \cup B \subseteq U$. □

C. Potential Recovery Lines

We now apply Lemmas 5 and 6 to predicting the possible intersections of a given checkpoint graph G and all its potential recovery lines. An *operational session* [9] is the interval between the start of normal execution and the instance of error recovery. The entire program execution can be viewed as consisting of several operational sessions ordered by *session numbers*. By repeatedly applying Lemma 5 within the same operational session and Lemma 6 across consecutive sessions, every potential recovery line of G can be transformed to the recovery line of one of the graphs $\hat{G} - W$, $W \subseteq U$.

THEOREM 1 *Given a checkpoint graph $G = (V, E)$ and $v \in V$,*

$$v \in M^*(G') \text{ for some } G' = (V', E') \in \mathcal{G}_f(G)$$

if and only if

$$v \in M^*(\hat{G} - W) \text{ for some } W \subseteq U.$$

Proof. The *if* part is trivial because $\hat{G} - W \in \mathcal{G}_f(G)$. We now prove the *only if* part. If $v \in M^*(G')$ for some $G' \in \mathcal{G}_f(G)$, without loss of generality, we may assume G is in the kth session and G' belongs to the lth session where $l \geq k$. Let G_i denote the checkpoint graph at the end of the ith session and G_i^- denote the checkpoint graph as the ith session starts, i.e., immediately after the recovery or at the beginning of program execution. Clearly, $G_i \in \mathcal{G}_s(G_i^-)$. Also let U_i and U_i^- denote the set of n_i's for G_i and G_i^-, respectively. Now consider the graphs G_l^- and G_k, and a vertex u which exists during the evolution from G_k to G_l^-. If $u \in M^*(\hat{G}_l^- - W_l^-)$ for some $W_l^- \subseteq U_l^-$, $u \in M^*(\hat{G}_{l-1} - W_{l-1})$ for some $W_{l-1} \subseteq U_{l-1}$ by Lemma 6. Since $\hat{G}_{l-1} - W_{l-1} \in \mathcal{G}_s(G_{l-1}) \subseteq \mathcal{G}_s(G_{l-1}^-)$, $u \in M^*(\hat{G}_{l-1}^- - W_{l-1}^-)$ for some $W_{l-1}^- \subseteq U_{l-1}^-$ by Lemma 5. By induction, we have $u \in M^*(\hat{G}_k - W_k)$ for some $W_k \subseteq U_k$. Since $G' \in \mathcal{G}_s(G_l^-)$, $v \in M^*(\hat{G}_l^- - W_l^-)$ for some $W_l^- \subseteq U_l^-$ by Lemma 5. By the above induction result, $v \in M^*(\hat{G}_k - W_k)$ for some $W_k \subseteq U_k$. Finally by Lemma 5 again, we have $v \in M^*(\hat{G} - W)$ for some $W \subseteq U$ because $\hat{G}_k - W_k \in \mathcal{G}_s(G_k) \subseteq \mathcal{G}_s(G)$. □

Theorem 1 shows that if any checkpoint on a given checkpoint graph G may be useful for future recovery, it must belong to the recovery line of one of the 2^N "immediate supergraphs" of G. Therefore, if we apply the rollback propagation algorithm to each of the 2^N graphs $\hat{G} - W$, $W \subseteq U$, and take the union of all the resulting recovery lines, we can obtain the set of non-discardable checkpoints. However, this is an exponential algorithm and may be unacceptable for applications with a large number of processors. The next section describes how this complexity can further be reduced.

D. Deleting Vertices for Discarded Checkpoints

There is in fact another situation where existing vertices may be deleted. Once a checkpoint is determined to be discardable and its space is reclaimed, the corresponding vertex on the checkpoint graph can be deleted. However, since the deletion is not part of the program execution as in the recovery process, we can not simply remove all the edges connected to the deleted vertex. The deletion of a discarded checkpoint v must follow the procedures described below in order to preserve the relations implied through v among the remaining vertices.

1. A new edge is generated for each pair of incoming and outgoing edges of v.
2. The source vertices of all the incoming edges of v have to be remembered. When an outgoing edge of v is added in the future, it is replaced by outgoing edges from these source vertices.

Since none of the potential recovery lines can contain v and the relations among all the remaining vertices and the new vertices remain unchanged, the deletion of v will not affect any potential recovery line.

V PREDICTIVE CHECKPOINT SPACE RECLAMATION

By applying Lemma 1, we will show that each of the 2^N MM-chains in Theorem 1 can be "synthesized" from the same set of N MM-chains. An efficient algorithm is then developed for finding the set of non-discardable checkpoints.

LEMMA 7 *Given a poset $P = (S, <)$ and $A, B \subseteq S$,*

$$min(A \cup B) = min(min(A) \cup B).$$

Proof. Let $min'(A) = A \setminus min(A)$. For each $a' \in min'(A)$, there exists $a \in min(A)$ such that $a < a'$. Since $a,\ a' \in A \cup B$, $a' \notin min(A \cup B)$. Also, for each non-minimal $c' \in A \cup B$, there exists $c \in min(A \cup B)$, $c \notin min'(A)$ such that $c < c'$. Therefore,

$$min(A \cup B) = min(min(A) \cup min'(A) \cup B) = min(min(A) \cup B).$$

□

LEMMA 8 *Given a poset P, $M \in \mathcal{M}(P)$ and $M \preceq M_i \in \mathcal{M}(P)$ for $i \in [0, k-1]$ for any finite k. Define*

$$\bigwedge_{i \in [0,k-1]} M_i = (...((M_0 \wedge M_1) \wedge M_2) \ ... \ \wedge M_{k-1},$$

then

(a)

$$M \preceq \bigwedge_{i \in [0,k-1]} M_i \ \in \mathcal{M}(P);$$

(b)

$$\bigwedge_{i \in [0,k-1]} M_i = min(\bigcup_{i \in [0,k-1]} M_i).$$

Proof. Both parts will be proved by induction on k.

(a) By Lemma 1, $\mathcal{M}(P)$ is a lattice and so $M_0 \wedge M_1 \in \mathcal{M}(P)$. 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 it is true for $k = n - 1$, i.e.,

$$M \preceq \bigwedge_{i \in [0,n-2]} M_i \ \in \mathcal{M}(P). \tag{1}$$

Again, by Lemma 1,

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

Eq. (1) and $M \preceq M_{n-1}$ implies

$$M \preceq \bigwedge_{i \in [0,n-1]} M_i.$$

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

(b) The case $k = 2$ is true by Lemma 1. Assume it is true for $k = n - 1$, i.e.,

$$\bigwedge_{i\in[0,n-2]} M_i = min(\bigcup_{i\in[0,n-2]} M_i). \tag{2}$$

Applying part (a), Eq. (2) and Lemma 1, we have

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

Lemma 7 further gives that

$$min(min(\bigcup_{i\in[0,n-2]} M_i) \cup M_{n-1}) = min(\bigcup_{i\in[0,n-2]} M_i \cup M_{n-1}) = min(\bigcup_{i\in[0,n-1]} M_i).$$

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

LEMMA 9 *For every $W \subseteq U$,*

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

Proof. If there are k vertices in the set W, without loss of generality, we may assume they are $z_0, z_1, ..., z_{k-1}$, i.e., $\{n_i : n_i \in W\} = \{z_j : j \in [0, k-1]\}$. Since $\hat{G} - z_j \in \mathcal{G}_s(\hat{G} - W)$, $M^*(\hat{G} - W) \preceq M^*(\hat{G} - z_j)$ for all $j \in [0, k-1]$ by Lemma 4(e).

Now consider the graph $\hat{G}$. $\hat{G} \in \mathcal{G}_s(\hat{G} - z_j)$ implies that $M^*(\hat{G} - z_j) \in \mathcal{M}(\hat{G})$ for $j \in [0, k-1]$. Similarly, $M^*(\hat{G} - W) \in \mathcal{M}(\hat{G})$. By Lemma 8(a) and (b),

$$M^*(\hat{G} - W) \preceq \bigwedge_{j\in[0,k-1]} M^*(\hat{G} - z_j) = min(\bigcup_{j\in[0,k-1]} M^*(\hat{G} - z_j)) \in \mathcal{M}(\hat{G}). \tag{3}$$

Moreover, for every $j \in [0, k-1]$, there exists $u \in M^*(\hat{G} - z_j)$ with $p(u) = p(z_j)$ such that $u < z_j$. Since $u \in \bigcup_{j\in[0,k-1]} M^*(\hat{G} - z_j)$, $z_j \notin min(\bigcup_{j\in[0,k-1]} M^*(\hat{G} - z_j))$. We have, by Lemma 4(d), $min(\bigcup_{j\in[0,k-1]} M^*(\hat{G} - z_j)) \in \mathcal{M}(\hat{G} - W)$ and therefore

$$min(\bigcup_{j\in[0,k-1]} M^*(\hat{G} - z_j)) \preceq M^*(\hat{G} - W).$$

Together with Eq. (3), we have proved

$$M^*(\hat{G} - W) = min(\bigcup_{j\in[0,k-1]} M^*(\hat{G} - z_j)) = min(\bigcup_{n_i\in W} M^*(\hat{G} - n_i)).$$

□

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

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

Let $N_D(G)$ denote the set of non-discardable checkpoints of G. Theorem 2 states a major result of this paper in that it provides the basis for an efficient algorithm to find all non-discardable checkpoints.

THEOREM 2 *Given a checkpoint graph $G = (V, E)$,*

$$N_D(G) = \bigcup_{i\in[0,N-1]} M^*(\hat{G} - n_i) \cap V$$

Proof. For any $v \in \bigcup_{i\in[0,N-1]} M^*(\hat{G} - n_i) \cap V$, $v \in M^*(\hat{G} - n_i)$ for some $i \in [0, N-1]$. Since $\hat{G} - n_i \in \mathcal{G}_f(G)$, $v \in N_D(G)$ by definition. Thus $\bigcup_{i\in[0,N-1]} M^*(\hat{G} - n_i) \cap V \subseteq N_D(G)$.

Conversely, for any $v \in N_D(G)$, $v \in V$ and $v \in M^*(\hat{G} - W)$ for some $W \subseteq U$ by Theorem 1. Lemma 9 further gives that

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

Therefore, $N_D(G) \subseteq \bigcup_{i\in[0,N-1]} M^*(\hat{G} - n_i) \cap V$ and so we have

$$N_D(G) = \bigcup_{i\in[0,N-1]} M^*(\hat{G} - n_i) \cap V.$$

□

Based on Theorem 2 we now present the *Predictive Checkpoint Space Reclamation (PCSR)* algorithm in Fig. 8. The algorithm is of complexity $\mathbf{O}(N|E|)$, where $|E|$ is the total number of edges in the checkpoint graph.

/* N is the number of processors */
/* $\hat{G}$ and n_i are as defined in Fig. 5 */
for each $i \in [0, N-1]$ {
    apply the rollback propagation algorithm to the checkpoint graph $\hat{G} - n_i$ to find the recovery line;
    the checkpoints in the intersection of G and the recovery line are included in the set $N_D(G)$;
}
all the checkpoints not in $N_D(G)$ can be reclaimed.

Figure 8: The Predictive Checkpoint Space Reclamation Algorithm.

An example illustrating the execution of the PCSR algorithm on the checkpoint graph in Fig. 2(b) is shown in Fig. 9. The traditional checkpoint space reclamation algorithm can only reclaim the first checkpoint of each processor. The PCSR algorithm, however, determines that all the checkpoints marked "X" are discardable.

VI THE MAXIMUM NUMBER OF NON-DISCARDABLE CHECKPOINTS

Traditionally, only obsolete checkpoints can be discarded. Since it is possible for the domino effect to persist during program execution, a common perception is that a large number of non-obsolete checkpoints may have to be kept and the space overhead may constantly grow as a program proceeds. In a sense, this is a more serious disadvantage than slower recovery due to the domino effect because it results in unpredictable space overhead during normal execution. Theorem 2 not only identifies the minimum set of non-discardable checkpoints but also places an upper bound N^2 on the number of non-discardable checkpoints for a general checkpoint graph because each $M^*(\hat{G} - n_i)$, $i \in [0, N-1]$, consists of N checkpoints. A tighter upper bound obviously exists based on the following observation:

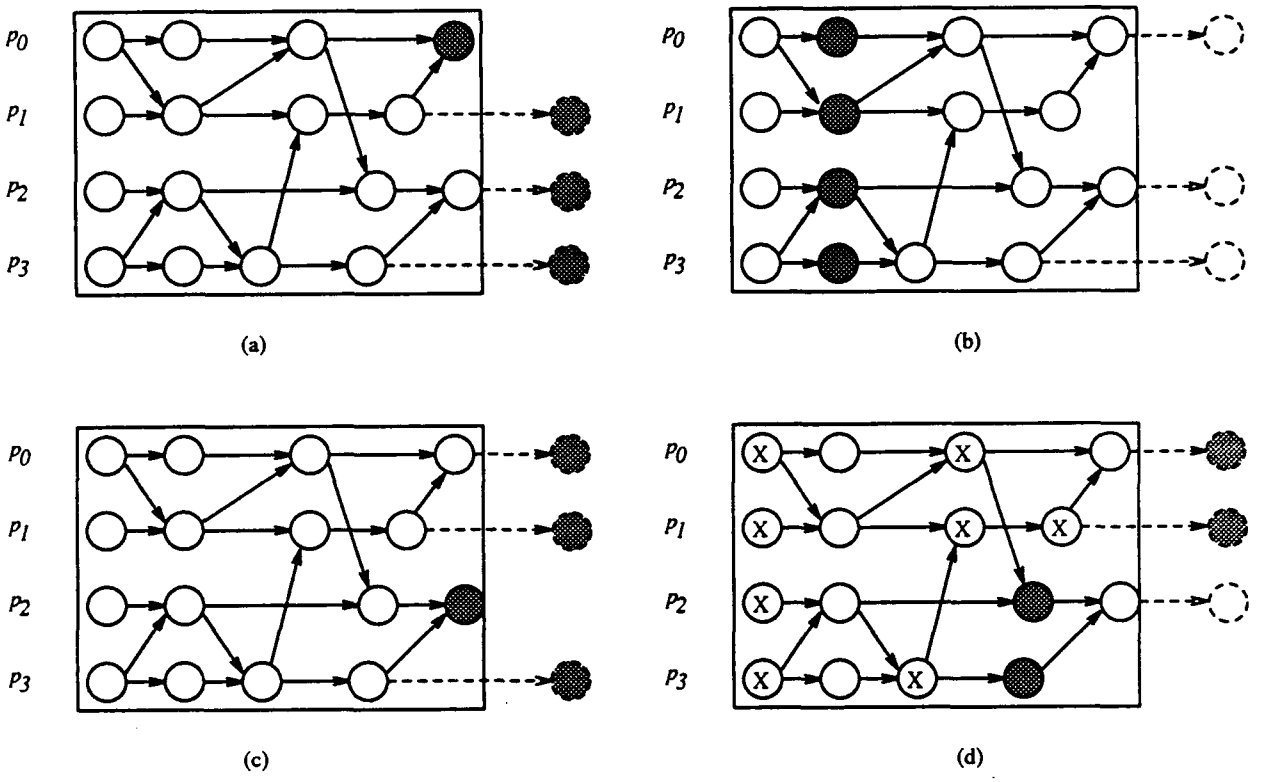

Figure 9: Execution of the PCSR algorithm (a) $\hat{G} - n_0$ (b) $\hat{G} - n_1$ (c) $\hat{G} - n_2$ (d) $\hat{G} - n_3$. (Shaded checkpoints belong to the recovery lines and checkpoints marked "X" are discardable.)

1. $M^*(\hat{G} - n_i)$ may contain vertices from the set U, but we are only concerned about vertices in G;

2. $M^*(\hat{G} - n_i)$'s may not be mutually disjoint;

3. if the last vertex e_i on chain C_i is maximal, $M^*(\hat{G} - n_i)$ will contribute only a single vertex to the set $N_D(G)$, i.e., e_i itself.

The following property addresses the implicit relations among $M^*(\hat{G} - n_i)$'s.

PROPERTY 3 *Let m_{ij} denote the vertex in $M^*(\hat{G} - n_i)$ from processor p_j. For $i, j \in [0, N-1]$ and $i \neq j$, if $m_{ij} \neq n_j$ and $m_{ji} \neq n_i$, then $M^*(\hat{G} - n_i) = M^*(\hat{G} - n_j)$.*

Proof. $m_{ij} \neq n_j$ implies $M^*(\hat{G}-n_i) \subseteq \hat{G}-n_i-n_j$. By Lemma 4(d) and (c), $M^*(\hat{G}-n_i) \in \mathcal{M}(\hat{G}-n_i-n_j) \subseteq \mathcal{M}(\hat{G}-n_j)$ and so

$$M^*(\hat{G}-n_i) \preceq M^*(\hat{G}-n_j).$$

Similarly, $m_{ji} \neq n_i$ leads to

$$M^*(\hat{G}-n_j) \preceq M^*(\hat{G}-n_i).$$

Since $(\mathcal{M}(\hat{G}-n_i-n_j), \preceq)$ forms a poset (Lemma 1(b)), we have

$$M^*(\hat{G}-n_i) = M^*(\hat{G}-n_j).$$

□

It should be noted that the PCSR algorithm can be further improved by applying Property 3. Inside the loop in Fig. 8, suppose we have found the recovery line $M^*(\hat{G}-n_i)$. Define the index set Γ as

$$\Gamma = \{j : m_{ij} \neq n_j,\ j \in [0, N-1] \text{ and } j > i\}.$$

Then for each later loop index $j \in \Gamma$, the rollback propagation algorithm can be aborted when any checkpoint from processor p_i is marked. Because this would mean $m_{ji} \neq n_i$ and $M^*(\hat{G}-n_j)$ is exactly the same as $M^*(\hat{G}-n_i)$.

THEOREM 3 *For any checkpoint graph $G = (V, E)$ in a system with N processors,*

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

Proof. By Theorem 2, we only have to consider the N^2 vertices m_{ij}, $i, j \in [0, N-1]$. For each $i \in [0, N-1]$, $m_{ii} \in V$ and contributes one vertex to $N_D(G)$. Since all the m_{ii}'s come from different processors, $N_D(G)$ now consists of N vertices. For the remaining $N^2 - N$ vertices with $i \neq j$, we consider each pair m_{ij} and m_{ji} at a time and there are $(N^2-N)/2$ such pairs. We distinguish three cases:

Case 1: $m_{ij} = n_j$ and $m_{ji} = n_i$. Both m_{ij} and m_{ji} do not belong to $N_D(G)$.

Case 2: $m_{ij} = n_j$ and $m_{ji} \neq n_i$, or $m_{ij} \neq n_j$ and $m_{ji} = n_i$. This pair will possibly add one new vertex to $N_D(G)$.

Case 3: $m_{ij} \neq n_j$ and $m_{ji} \neq n_i$. It follows that $M^*(\hat{G} - n_i) = M^*(\hat{G} - n_j)$ by Property 3, and so $m_{ij} = m_{jj}$ and $m_{ji} = m_{ii}$. Since m_{jj} and m_{ii} are already in $N_D(G)$, this case does not increase the size of $N_D(G)$.

Therefore, each of the $(N^2 - N)/2$ pairs can contribute at most one new vertex to $N_D(G)$. We then have

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

□

One may argue that the upper bound derived in Theorem 3 is still of order N^2. We will next show that $N(N+1)/2$ is in fact the lowest upper bound, i.e., the maximum, because for any N we can construct a checkpoint graph, G_N^*, as shown in Fig. 10 to achieve this upper bound. By applying the PCSR algorithm shown in Fig. 8, it is not hard to see that all the $N(N+1)/2$ vertices in Fig. 10 are non-discardable.

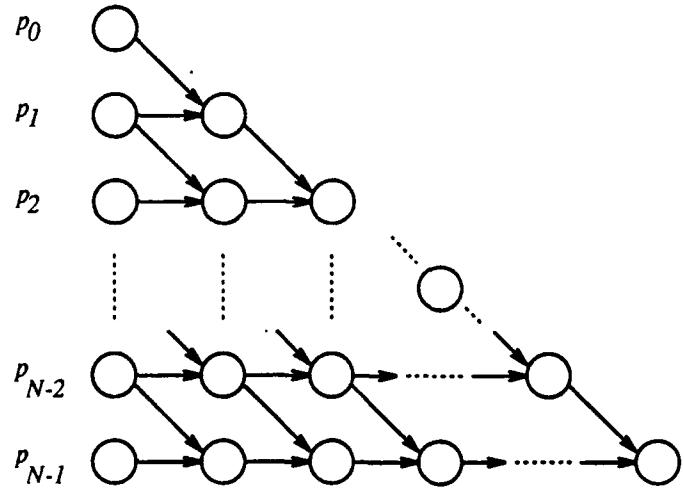

Figure 10: G_N^*: The checkpoint graph with $N(N+1)/2$ non-discardable checkpoints.

When a checkpoint graph is given, we can further reduce the maximum by counting the number, L, of maximal vertices in the set of e_i's (Fig. 5). Recall that if e_i is maximal, $m_{ii} = e_i$ and $m_{ij} = n_j$ for $j \neq i$. Therefore, in the discussion for each pair of m_{ij} and m_{ji} in the proof of Theorem 3, the case when both e_i and e_j are maximal corresponds to Case 1. The maximum then becomes

$$|N_D(G)| \leq N + \binom{N}{2} - \binom{L}{2}.$$

In particular when $L = N$, $|N_D(G)| = N$, which corresponds to the case of coordinated checkpointing.

VII EXPERIMENTAL RESULTS

Four parallel programs are used to illustrate the checkpoint space reclamation capabilities and benefits of the PCSR algorithm. Two are CAD programs written for the Intel iPSC/2 hypercube: a Cell placement program and a Channel router program. The other two are the Knight tour program and the N-queen program written in the *Chare Kernel* language which has been developed as a medium-grain, message-driven and machine-independent parallel language [24]. We use the version of Chare Kernel on the Encore Multimax 510 multiprocessor. The periodic checkpointing routine is implemented as the interrupt service routine for UNIX *alarm(T)* system call, where T is the checkpoint interval in seconds. Each processor sets the alarm at the very beginning of the program and the checkpointing routine independently. A concurrent checkpointing algorithm as described by Li et. al [25] is assumed so that the program thread is interrupted for a small, fixed amount of time (0.1 seconds) for taking each checkpoint, after which the checkpointing thread executes concurrently with the program thread to finish the checkpointing. Communication traces are collected by intercepting the "send" and "receive" system calls. Communication trace-driven simulation is then performed to obtain the results.

The number of processors used and the total execution time for each program are listed in Table 1. The checkpoint interval for each program is arbitrarily chosen to be approximately one tenth of the execution time.

Table 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	442.0	273.2	1625.1
Checkpoint interval (sec)	35	40	30	150

Figs. 11-14 compares our PCSR algorithm with the traditional checkpoint space reclamation algorithm for typical executions of the four programs. Since obsolete checkpoints must be discardable, the curves for non-discardable checkpoints are always below the curves for non-obsolete checkpoints. Note that the curves do not show the actual number of checkpoints that would be kept on stable storage during program execution because the checkpoint space reclamation algorithm would not be continuously active throughout the program execution. Instead, it shows the number of checkpoints which have to be kept 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 non-obsolete checkpoints as the total number of checkpoints increases. These figures show that the PCSR algorithm is particularly effective when the domino effect persists. The largest difference between the number of non-obsolete checkpoints and the number of non-discardable checkpoints for each figure is: 39 versus 7 for Cell placement, 40 versus 12 for Channel router, 24 versus 10 for Knight tour and 41 versus 5 for N-queen.

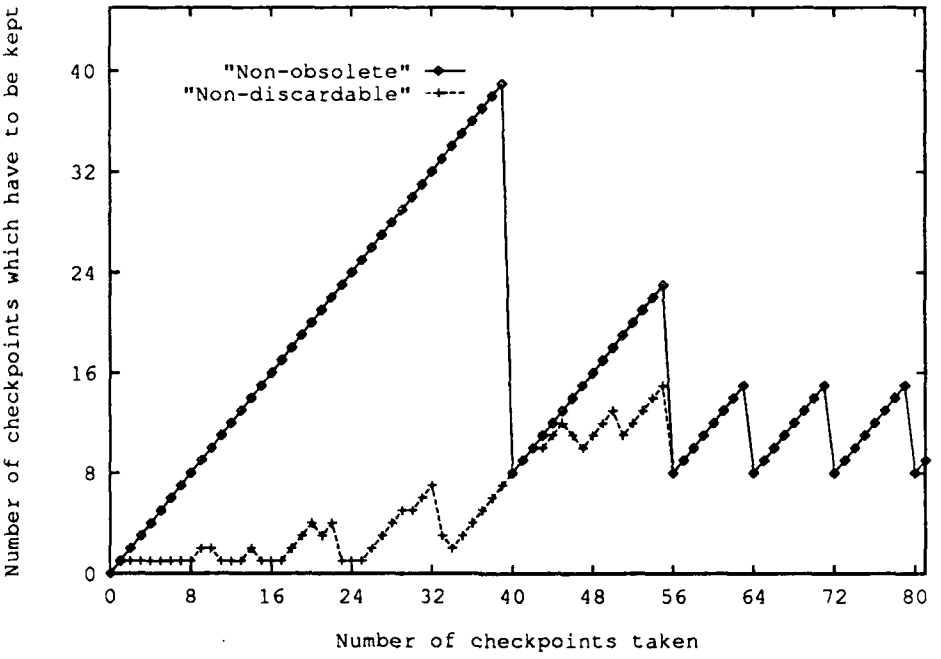

Figure 11: Non-obsolete checkpoints and non-discardable checkpoints for the Cell placement program.

VIII CONCLUSIONS

The problem of finding recovery lines for independent checkpointing schemes was formulated as determining the maximal maximum-sized antichains of partially ordered sets. We presented a method for predicting the possibility of any checkpoint becoming a member of future recovery line, and showed that sometimes checkpoints will never be needed for recovery so their space can be reclaimed. Based on the algorithm for finding the recovery lines, a new checkpoint space reclamation algorithm, with complexity linear in the number of processors N and linear in the number of edges in the checkpoint graph, was developed for determining the set of non-discardable checkpoints. The maximum, $N(N+1)/2$, on the number of non-discardable checkpoints for an arbitrary checkpoint graph was also derived to show that the space overhead for maintaining multiple checkpoints is bounded even when

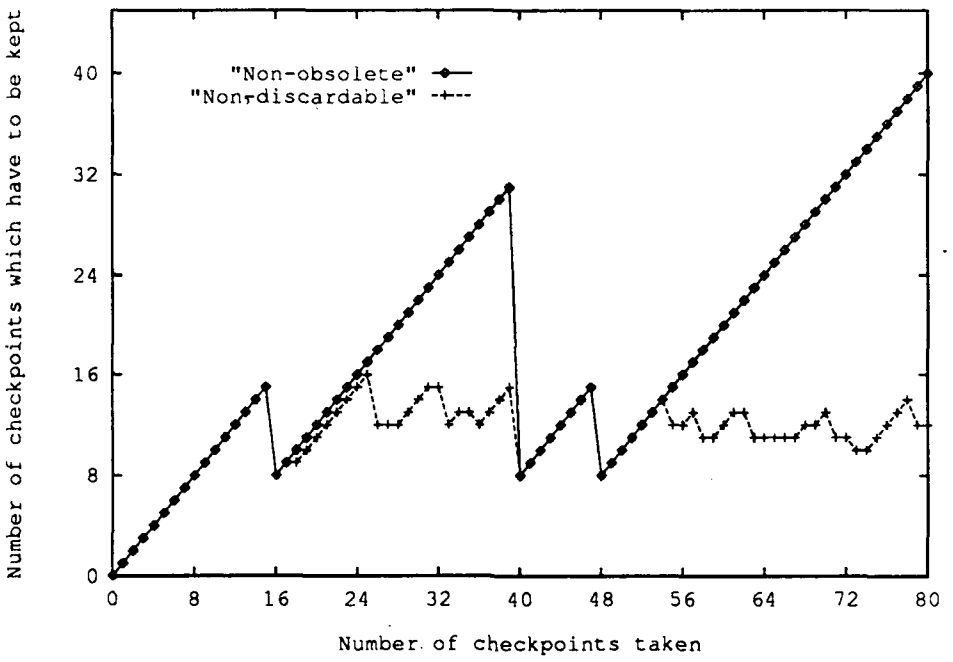

Figure 12: Non-obsolete checkpoints and non-discardable checkpoints for the Channel router program.

the domino effect persists during program execution. Communication trace-driven simulation for four parallel programs illustrated that the algorithm can be effective in significantly reducing the number of retained checkpoints.

ACKNOWLEDGEMENT

The authors wish to express their sincere thanks to Douglas West, Weiping Shi, Shyh-Kwei Chen and Sanjeev Khanna for their valuable discussions, to Balkrishna Ramkumar and Junsheng Long for their help with the experimental results, to L. V. Kalé for access to the Chare Kernel, and to Prith Banerjee for use of his hypercube programs.

REFERENCES

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

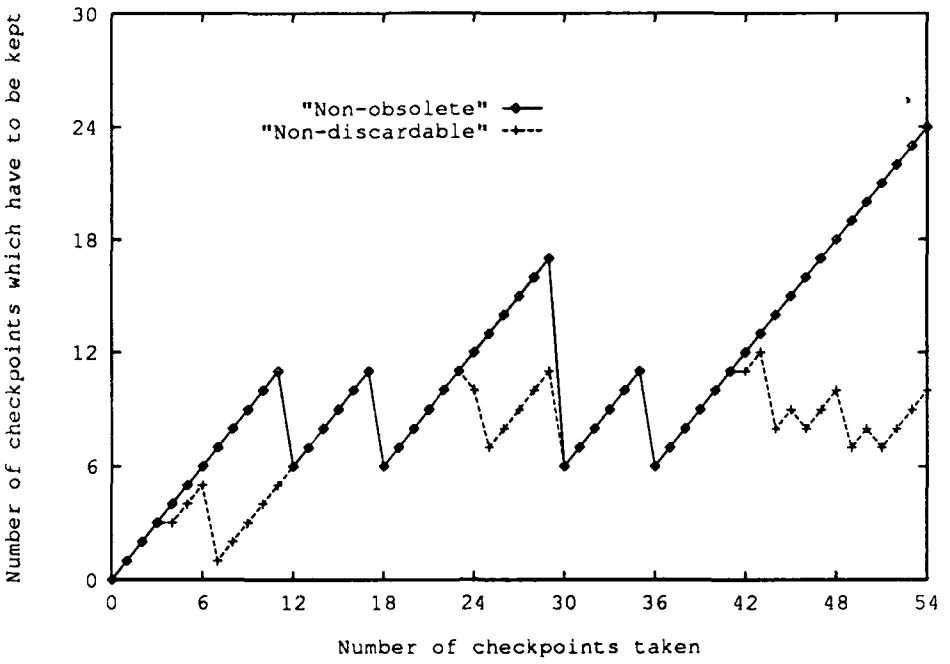

Figure 13: Non-obsolete checkpoints and non-discardable checkpoints for the Knight tour program.

[2] K. H. Kim and J. H. You, "A highly decentralized implementation model for the Programmer-Transparent Coordination (PTC) scheme for cooperative recovery," in *Proc. Symp. on Fault-Tolerant Computing*, pp. 282–289, 1990.

[3] Y. M. Wang and W. K. Fuchs, "Optimistic message logging for independent checkpointing in message-passing systems." Submitted to *IEEE Symp. on Reliable Distributed Systems*, 1992.

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

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

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

[7] 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*, pp. 124–130, 1981.

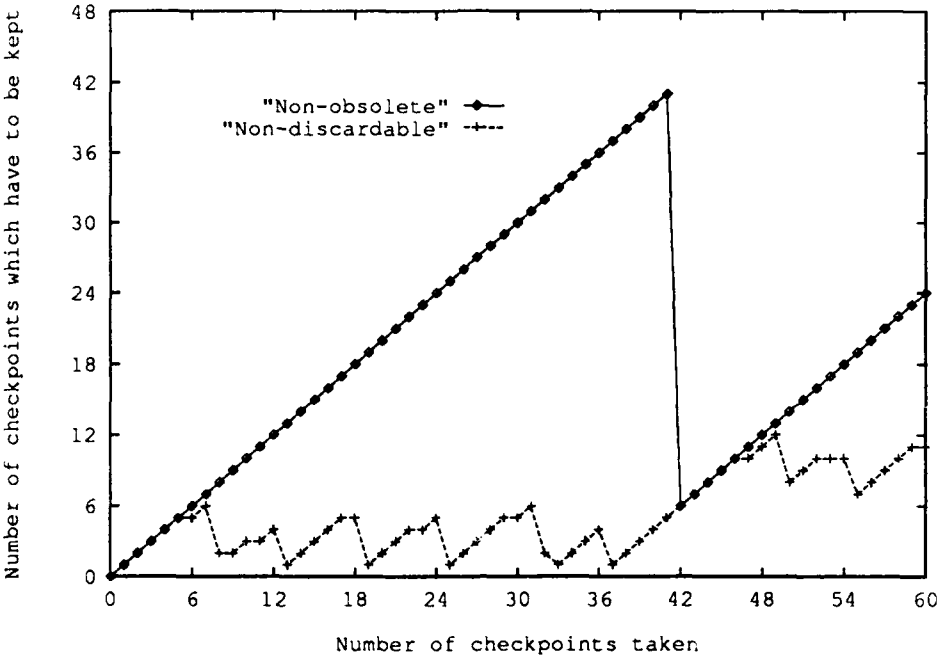

Figure 14: Non-obsolete checkpoints and non-discardable checkpoints for the N-queen program.

[8] R. E. Strom and S. Yemini, "Optimistic recovery in distributed systems," *ACM Trans. on Computer Systems*, vol. 3, pp. 204–226, Aug. 1985.

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

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

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

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

[13] D. B. Johnson and W. Zwaenepoel, "Transparent optimistic rollback recovery," *Operating Systems Review*, pp. 99–102, Apr. 1991.

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

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

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

[17] 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.

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

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

[20] K. P. Bogart, *Introductory combinatorics*. Pitman Publishing Inc., Massachusetts, 1983.

[21] I. Anderson, *Combinatorics of finite sets*. Clarendon Press, Oxford, 1987.

[22] R. P. Dilworth, "Some combinatorial problems on partially ordered sets," *Combinatorial Analysis, Proc. Symp. Appl. Math.*, vol. 10, pp. 85–90, 1960.

[23] D. B. West, "A polynomial-time algorithm for finding the maximal maximum-sized antichain." Personal communication, 1991.

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

[25] K. Li, J. F. Naughton, and J. S. Plank, "Real-time, concurrent checkpointing for parallel programs," in *Proc. 2nd ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pp. 79–88, Mar. 1990.