Tight Upper Bound on Useful Distributed System Checkpoints

Yi-Min Wang
AT&T Bell Laboratories
Murray Hill, New Jersey

Pi-Yu Chung
AT&T Bell Laboratories
Murray Hill, New Jersey

W. Kent Fuchs
University of Illinois
Urbana, Illinois

Abstract

In this paper, we give an alternative proof of the necessary and sufficient condition for achieving optimal checkpoint garbage collection in distributed systems [9]. We show that, by formulating the recovery line calculation problem as a reachability analysis problem on a rollback-dependency graph, the proof can be greatly simplified. We also present a polynomial-time optimal garbage collection algorithm based on a graph search.

Keywords: fault tolerance, distributed systems

1 Introduction

A checkpoint is a snapshot of process state, saved on nonvolatile storage to survive failures. A process periodically takes checkpoints during its execution so that when its volatile state is lost due to a failure, the execution can resume from a checkpointed state (an action called *rollback recovery*) instead of from the very beginning. In a distributed system, two checkpoints c_1 and c_2 of two processes p_1 and p_2 are *inconsistent* if a message was sent from p_1 after c_1 and received by p_2 before c_2, i.e., c_1 *happened-before* c_2 [3, 7], or vice versa. When a failure occurs, the unique best consistent set of states which minimizes the amount of rollbacks[1] needs to be calculated based on the recorded dependencies so that the state of the entire system can be restored to a state that could have happened. In this paper, we refer to such a set as the **recovery line**.

The purpose of garbage collection is to discard those **garbage checkpoints** that can never be useful for any recovery in order to reclaim the nonvolatile storage space they occupied. A garbage collection algorithm is *optimal* if it can discard all garbage checkpoints, and any checkpoint that it does not discard must be a **useful checkpoint** for some future recovery. Optimal garbage collection is a hard problem because one in general cannot predict future message dependencies, checkpoints and recoveries. Over the past decade, a simple sufficient condition based on the notion of *obsolete checkpoints* has been used [1, 5, 6]: the most recent set of consistent checkpoints, $\mathcal{C}$, is calculated; all the checkpoints taken before $\mathcal{C}$ are obsolete and can be discarded; all nonobsolete checkpoints taken after $\mathcal{C}$ (including checkpoints of $\mathcal{C}$) are retained because they may be combined with some future checkpoints to form a better set of consistent checkpoints than $\mathcal{C}$. If processes are allowed to take their checkpoints independently without any coordination with each other, it has been shown that [4] $\mathcal{C}$ may always consist of the same set of old checkpoints no matter how many new checkpoints have been taken. Therefore, the number of nonobsolete checkpoints and hence the space overhead for storing them are unbounded.

The major challenge of the optimal garbage collection problem is that it requires the consideration of an infinite number of possible future failure scenarios, including an arbitrary number of recoveries. The main contribution of our work is to prove that it suffices to consider a finite number of *immediate failure scenarios* [9]. We formulate recovery

[1] More generally, the unique set minimizes all *reasonable cost functions* [2].

line calculation as a reachability analysis problem on a *rollback dependency graph* [8]. Given a current graph G, we prove that every checkpoint useful for a future recovery must also be useful for one of the 2^N immediate failure scenarios where N is the number of processes and each of the 2^N scenarios is defined by the failure of a particular subset of the N processes. We further show that N out of the 2^N recovery lines contain all of the checkpoints in the 2^N recovery lines. The optimal garbage collection algorithm is therefore a polynomial-time algorithm for calculating these N recovery lines, and the number of useful checkpoints is therefore bounded by N^2 since each recovery line contains at most N checkpoints. By exploiting an inherent constraint among the N recovery lines, we refine the bound to be $N(N+1)/2$ and prove that it is tight by constructing a worse-case graph to achieve the bound, given any N.

2 Models and Protocols

Checkpointing

Let N be the number of processes in a distributed system. Let $c_{i,x}$, $0 \leq i \leq N-1$, $0 \leq x$, denote the xth checkpoint of process p_i. (Checkpoint $c_{i,0}$ represents the state before p_i's execution.) The *checkpoint interval* between $c_{i,x-1}$ and $c_{i,x}$ is the xth checkpoint interval of p_i and is denoted by $I_{i,x}$. When a message is sent from $I_{i,x}$, it is tagged with the pair of integers i and x so that the dependency can be tracked at the receiver side. We define a **rollback dependency graph** (or **R-graph**) in which each node represents a checkpoint. If there exists a message m sent from $I_{i,x}$ and received in $I_{j,y}$, an edge is drawn from node $c_{i,x}$ to node $c_{j,y}$. Basically, such an edge indicates that if p_i rolls back to a state before $c_{i,x}$, then the effect of m should be undone and so p_j needs to roll back to a state before $c_{j,y}$. When checkpoint $c_{j,y}$ is taken, the process state of p_j and all the recorded incoming edges of node $c_{j,y}$ are sent to a central nonvolatile storage server.

Rollback recovery

When a subset of processes fails, the central server constructs an R-graph to calculate the recovery line. In addition to using the **nonvolatile checkpoints**, the server treats the volatile states of surviving processes as **volatile checkpoints** and requests the up-to-date dependency information to be included in the graph. Figure 1(a)-(c) give an example of recovery line calculation when p_0 and p_1 fail. The volatile checkpoints of p_0 and p_1 are *initially marked* to indicate the fact that they have to be rolled back, i.e., p_0 and p_1 have to roll back to some checkpointed states before them because the failure has destroyed the volatile states. A search is performed starting from the initially marked nodes, and all the reachable nodes are also marked to indicate that their corresponding checkpoints should also be rolled back due to message dependencies. It is not hard to see that, after the search, **the set of the last unmarked node of each process** forms the recovery line. We will refer to the above algorithm as the *recovery line algorithm*. In the Figure 1(b) example, checkpoints $c_{0,2}$ and $c_{1,2}$ belong to the recovery line, which means p_0 and p_1 need to roll back to the states saved in $c_{0,2}$ and $c_{1,2}$, respectively. Processes p_2 and p_3 can simply continue their executions without any rollbacks because their volatile checkpoints are on the recovery line. Figure 1(c) shows the R-graph immediately after the recovery.

Garbage collection

The central server periodically executes a garbage collection algorithm in order to reclaim the storage space occupied by checkpoints that will never be useful. Garbage collection is performed based on the server's local

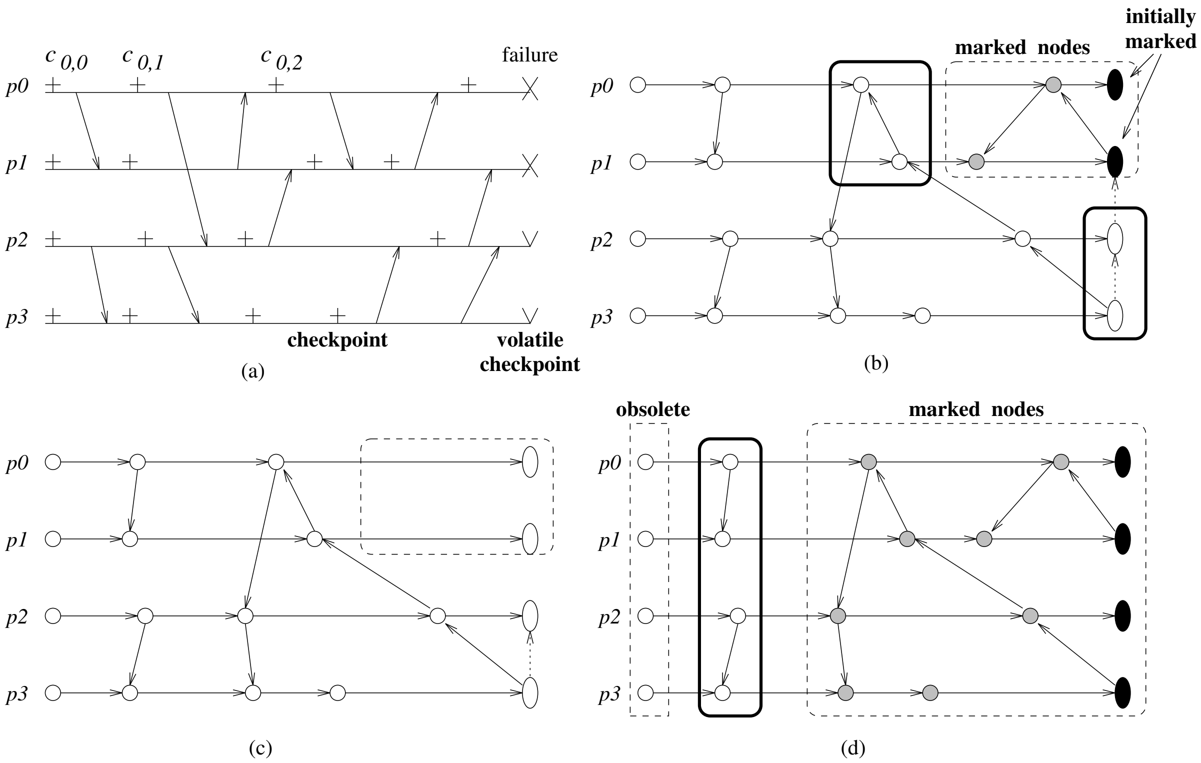

Figure 1: (a) Example checkpoint and message pattern; (b) R-graph and recovery line (thick solid line); (c) R-graph immediately after recovery; (d) nonvolatile R-graph for garbage collection.

nonvolatile dependency information. For example, suppose the failure in Figure 1(b) does not occur and a garbage collection algorithm is invoked. Figure 1(d) illustrates the **nonvolatile R-graph** which excludes the incoming edges of the volatile checkpoints (dotted edges) from the R-graph in (b). Traditional garbage collection algorithms work as follows: a search is started by initially marking all volatile checkpoints so that the calculated recovery line $\mathcal{C}$ involves only nonvolatile checkpoints; all the obsolete checkpoints before $\mathcal{C}$ can be discarded; all the nonobsolete checkpoints must be retained.

3 Optimal Garbage Collection and Tight Upper Bound

Given a nonvolatile R-graph, an optimal garbage collection algorithm must guarantee that any garbage checkpoint it discarded must not be useful for any future recovery, and any useful checkpoint it retained must be useful for some future recovery. A program execution can be modeled as consisting of an arbitrary number of alternating *normal sessions* and *recovery sessions*, as shown in Figure 2. In a normal session, new checkpoints are taken and new dependencies are recorded. A normal session ends and a recovery session starts when a rollback is initiated. In a recovery session, the recovery line is calculated and the checkpoints after the recovery line are removed from the R-graph. A recovery session ends and a new normal session starts when the system finishes its recovery by rolling back to the recovery line.

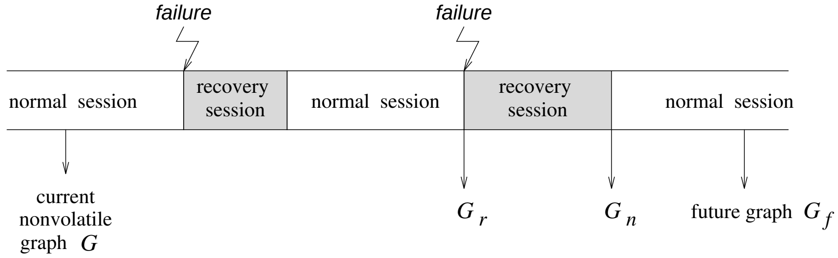

Figure 2: Program execution model: normal and recovery sessions.

Notations

The following notations will be used throughout the rest of the paper:

- $I(G,T)$: given an R-graph G, we define a set of 2^N *immediate recovery lines*, $I(G,T)$, each of which is obtained by initially marking a subset T of volatile checkpoints, i.e., $T \subseteq V$ where V denotes the set of all N volatile checkpoints;
- $R(T)$: the set of nodes that are reachable from a node in the set T;
- $T \rightarrow c$: node c can be reached from a node in the set T;
- $T \not\rightarrow c$: node c cannot be reached from any node in the set T;
- v_i: the volatile checkpoint of process p_i;
- $k_{i,x+1}$: the checkpoint of p_i immediately following $c_{i,x}$; $k_{i,x+1}$ can be a nonvolatile $c_{i,x+1}$ or a volatile v_i
- V_r: the set of all N volatile checkpoints of an R-graph G_r.

LEMMA 1: mapping from G_f to G_n

Let G_f be an R-graph in a normal session and G_n be the R-graph at the beginning of that session. Any nonvolatile checkpoint of G_n which appears in an immediate recovery line of G_f must also belong to an immediate recovery line of G_n. Formally, given $c_{i,x} \in I(G_f, T_f)$ for some $T_f \subseteq V_f$, if $c_{i,x}$ is a nonvolatile checkpoint of G_n, then $c_{i,x} \in I(G_n, T_n)$ for some $T_n \subseteq V_n$.

Proof. In G_f, given $c_{i,x} \in I(G_f, T_f)$ and $c_{i,x}$ is a nonvolatile checkpoint of G_n, we first partition $I(G_f, T_f) = C_1 \cup C_2$ where C_1 consists of nonvolatile checkpoints of G_n and $C_2 = I(G_f, T_f) \setminus C_1$; so $c_{i,x} \in C_1$. Then, corresponding to C_1 and C_2, we partition into $F_1 \cup F_2$ the set of checkpoints which immediately follows the last nonvolatile checkpoint of each process of G_n, as shown in Figure 3. Clearly, checkpoints in $F_1 \cup F_2$ can be nonvolatile or volatile, and they must have "evolved" from the volatile checkpoints of G_n and hence must contain the dependency edges that already existed in G_n. Also, $F_1 \subseteq R(T_f)$.

According to the recovery line algorithm, if $c_{i,x} \in I(G_f, T_f)$, then there exists $x \in T_f$ such that $x \not\rightarrow c_{i,x}$ and $x \rightarrow k_{i,x+1}$. Consider the case where $k_{i,x+1}$ is a nonvolatile checkpoint of G_n, i.e., $k_{i,x+1} = c_{i,x+1}$. By construction, all incoming edges of $c_{i,x+1}$ must have existed in G_n. Therefore, in order for x to reach $c_{i,x+1}$, the path $x \rightarrow c_{i,x+1}$ must go though some $y \in F_1 \cup F_2$. (If the path $x \rightarrow c_{i,x+1}$ contains multiple nodes from $F_1 \cup F_2$, we choose y to be the last one of them on the path.) Since x cannot reach any checkpoint in C_2, we must have $y \in F_1$. Also, the path $y \rightarrow c_{i,x+1}$ must have existed in G_n as $v \rightarrow c_{i,x+1}$ where v is a volatile checkpoint of G_n.

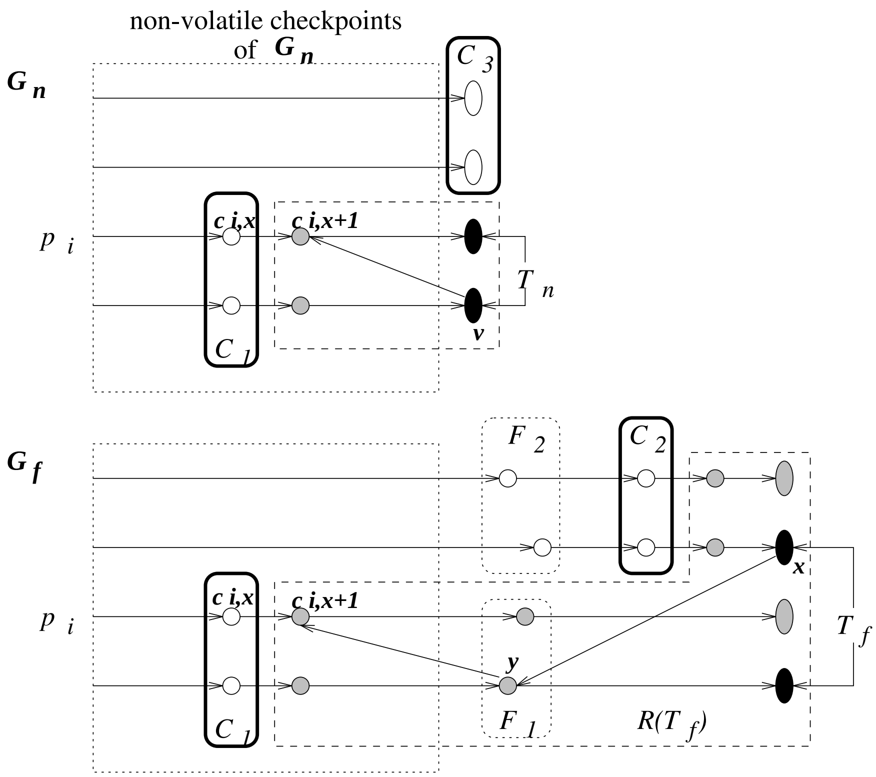

Figure 3: Lemma 1.

In G_n, let T_n denote the set of volatile checkpoints corresponding to F_1. We then have $v \in T_n$. Our goal is to show that $c_{i,x} \in I(G_n, T_n)$ by proving that $T_n \rightarrow k_{i,x+1}$ and $c_{i,x} \notin R(T_n)$. If $k_{i,x+1} = c_{i,x+1}$ is a nonvolatile checkpoint of G_n, then $v \rightarrow c_{i,x+1}$ leads to $T_n \rightarrow c_{i,x+1}$; otherwise, $k_{i,x+1} = v_i \in T_n$ and so $T_n \rightarrow v_i$ trivially. Therefore, $T_n \rightarrow k_{i,x+1}$ in G_n.

Since $c_{i,x} \in R(T_n)$ in G_n would imply $c_{i,x} \in R(F_1)$ in G_f which would lead to a contradiction $c_{i,x} \in R(T_f)$, we must have $c_{i,x} \notin R(T_n)$ in G_n. Therefore, we have proved that $c_{i,x} \in I(G_n, T_n)$. ■

LEMMA 2: mapping from G_n to G_r

Let G_n be the R-graph at the beginning of a normal session after a recovery, and G_r be the R-graph at the end of the previous normal session, as shown in Figure 4(a) and (b). Any nonvolatile checkpoint which appears in an immediate recovery line of G_n must also belong to an immediate recovery line of G_r. Formally, given a nonvolatile checkpoint $c_{i,x}$, if $c_{i,x} \in I(G_n, T_n)$ for some $T_n \subseteq V_n$, then $c_{i,x} \in I(G_r, T_r)$ for some $T_r \subseteq V_r$.

Proof. First, we partition $I(G_n, T_n) = C_1 \cup C_3$ where C_1 consists of nonvolatile checkpoints of G_n and $C_3 = I(G_n, T_n) \setminus C_1$. Clearly, the size of T_n is no greater than the size of C_1. If any process p_j has a nonvolatile checkpoint c_j in C_1 but its volatile checkpoint v_j is not in T_n, then we add v_j to T_n and call the new set T'_n. The recovery line $I(G_n, T'_n)$ is the same as $I(G_n, T_n)$ because the fact that T_n can reach at least one checkpoint of p_j implies that $T_n \rightarrow v_j$ and so $R(T'_n) = R(T_n) \cup R(v_j) = R(T_n)$. We repeat the same procedure for every such p_j

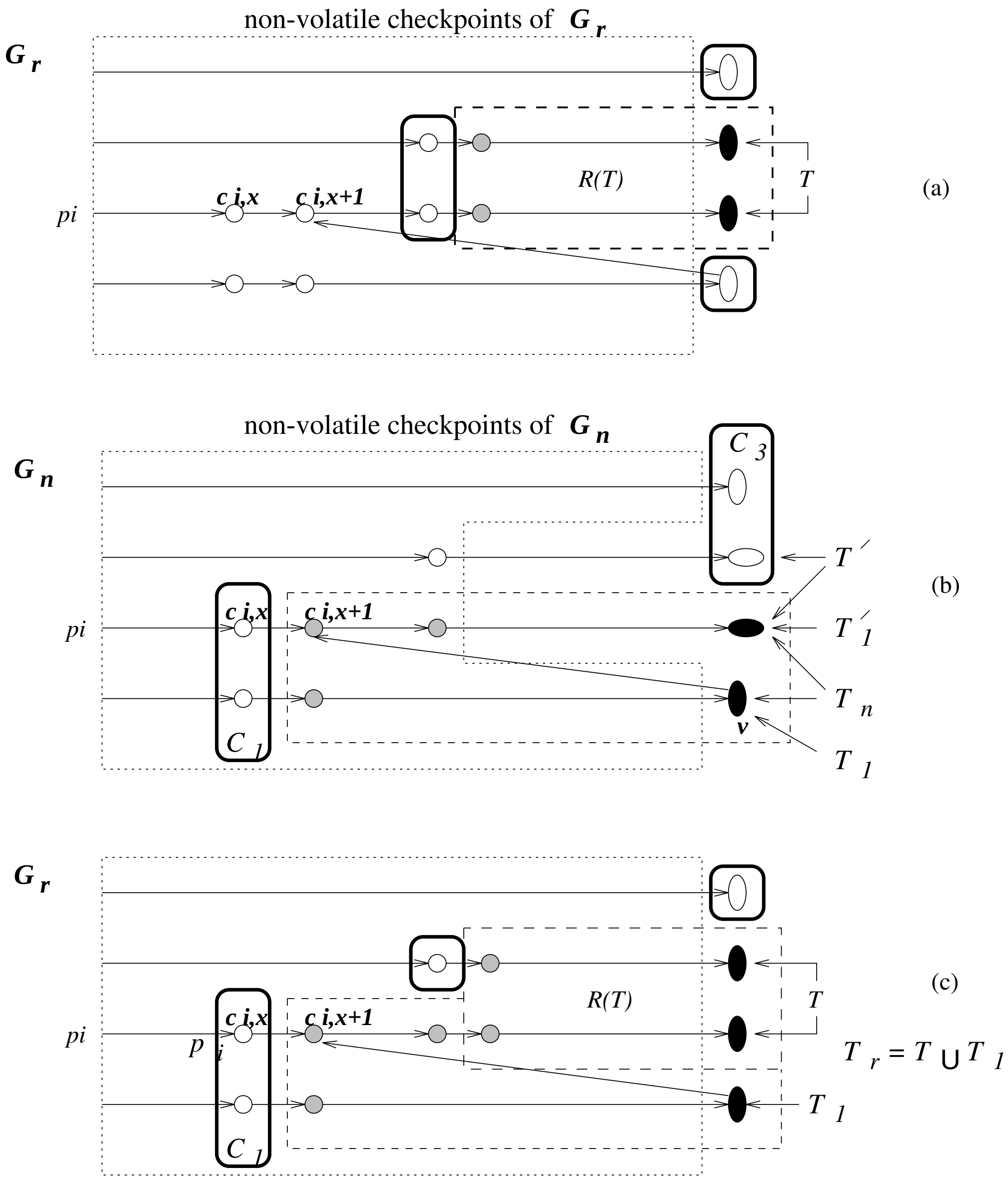

Figure 4: Lemma 2.

until the size of T_n' is the same as the size of C_1, i.e., until T_n' and C_1 span the same subset of processes. Since we do not change the recovery line, we will use such a T_n' as the new T_n in the proof.

Next, we analyze the relationship between G_r and G_n. Let T denote the set of initially marked volatile checkpoints of G_r which starts the recovery session that ends with G_n, as shown in Figure 4(a). The recovery line $I(G_r, T)$ is calculated and the set of checkpoints taken after the recovery line, i.e., $R(T)$, is removed from G_r. The remaining graph is G_n excluding the new set of volatile checkpoints, denoted by T' in Figure 4(b)[2], which is added to represent the volatile states of rolled-back processes immediately after the recovery. Every volatile checkpoint of T' has only one incoming edge and no outgoing edge because previous execution has been rolled back and new execution has not started yet.

Given a nonvolatile checkpoint $c_{i,x} \in I(G_n, T_n)$, $T_n \subseteq V_n$, as shown in Figure 4(b), we have $c_{i,x+1} \in R(T_n)$ and $c_{i,x} \notin R(T_n)$. Define $T_1 = T_n \setminus T'$ and $T_1' = T_n \setminus T_1$. Since, in G_n, volatile checkpoints in T' do not have any outgoing edge, we have $R(T_n) = R(T_1) \cup R(T_1') = R(T_1) \cup T_1'$. If $c_{i,x+1}$ is a nonvolatile checkpoint in G_n, then $c_{i,x+1} \notin T_1'$ and so $c_{i,x+1} \in R(T_1)$ (Case 1); otherwise, $c_{i,x}$ is p_i's last nonvolatile checkpoint in G_n (Case 2).

Now consider G_r in Figure 4(c). Our goal is to show that $c_{i,x} \in I(G_r, T \cup T_1)$ by proving that $k_{i,x+1} \in R(T \cup T_1)$ and $T \cup T_1 \not\rightarrow c_{i,x}$. In Case 1, $c_{i,x+1} \in R(T_1)$ remains true in G_r because all the edges in G_n also exist in G_r (except for the edges pointing to nodes in T'). In Case 2, if p_i is a rolled-back process, then $k_{i,x+1} \in R(T)$; otherwise, $k_{i,x+1}$ is a volatile checkpoint and we have $k_{i,x+1} \in T_1$ by the construction of T_n at the very beginning of the proof. As a result, we have $k_{i,x+1} \in R(T \cup T_1)$ for all cases.

We cannot have $T \rightarrow c_{i,x}$ because that would make $c_{i,x}$ part of $R(T)$. Recall that $T_1 \not\rightarrow c_{i,x}$ in G_n. In G_r, some nodes of T_1 may have additional edges pointing into $R(T)$. But since $R(T) \not\rightarrow c_{i,x}$, $T_1 \not\rightarrow c_{i,x}$ remains true in G_r. In summary, $T \cup T_1 \not\rightarrow c_{i,x}$ and thus we have $c_{i,x} \in I(G_r, T_r)$ where $T_r = T \cup T_1 \subseteq V_r$. ■

LEMMA 3: reduction from 2^N to N

Any nonvolatile checkpoint of G which appears in one of G's 2^N immediate recovery lines must also belong to an immediate recovery line obtained by initially marking only one volatile checkpoint. Formally, for any nonvolatile checkpoint $c_{i,x}$, if $c_{i,x} \in I(G, T), T \subseteq V$, then $c_{i,x} \in I(G, v)$, $v \in V$.

Proof. If $c_{i,x} \in I(G, T)$, then $T \rightarrow k_{i,x+1}$ and $T \not\rightarrow c_{i,x}$. Therefore, there exists $v \in T$ such that $v \rightarrow k_{i,x+1}$ and $v \not\rightarrow c_{i,x}$. We then have $c_{i,x} \in I(G, v)$ where $v \in T \subseteq V$. ■

THEOREM 1: optimal garbage collection

Given a nonvolatile R-graph G, the set of useful checkpoints is equivalent to the set of nonvolatile checkpoints in the union of the N immediate recovery lines $I(G, v)$, $v \in V$.

Proof. If a nonvolatile checkpoint c of G is useful, then by definition c must appear in an immediate recovery line of a future graph G_f, and c must exist in every R-graph at the boundary of sessions between G and G_f. By starting with G_f and repeatedly and alternately applying Lemma 1 and Lemma 2, we can show that c must belong to

[2]Volatile checkpoints in T' are drawn as horizontal ovals to distinguish from the corresponding volatile checkpoints in T.

an immediate recovery line of G. From Lemma 3, we have $c \in I(G, v)$, $v \in V$. Conversely, if $c \in I(G, v)$ for some $v \in V$, then clearly c is useful for an immediate failure recovery. ■

Figure 5 illustrates the optimal garbage collection for the nonvolatile R-graph shown in Figure 1(d). While traditional algorithms can only discard the very first four checkpoints, our algorithm identifies the eight useful checkpoints (as circled by the thick solid lines) and can discard the remaining eight garbage checkpoints. The complexity of the algorithm is $O(Nm)$ where m is the number of edges in a nonvolatile R-graph.

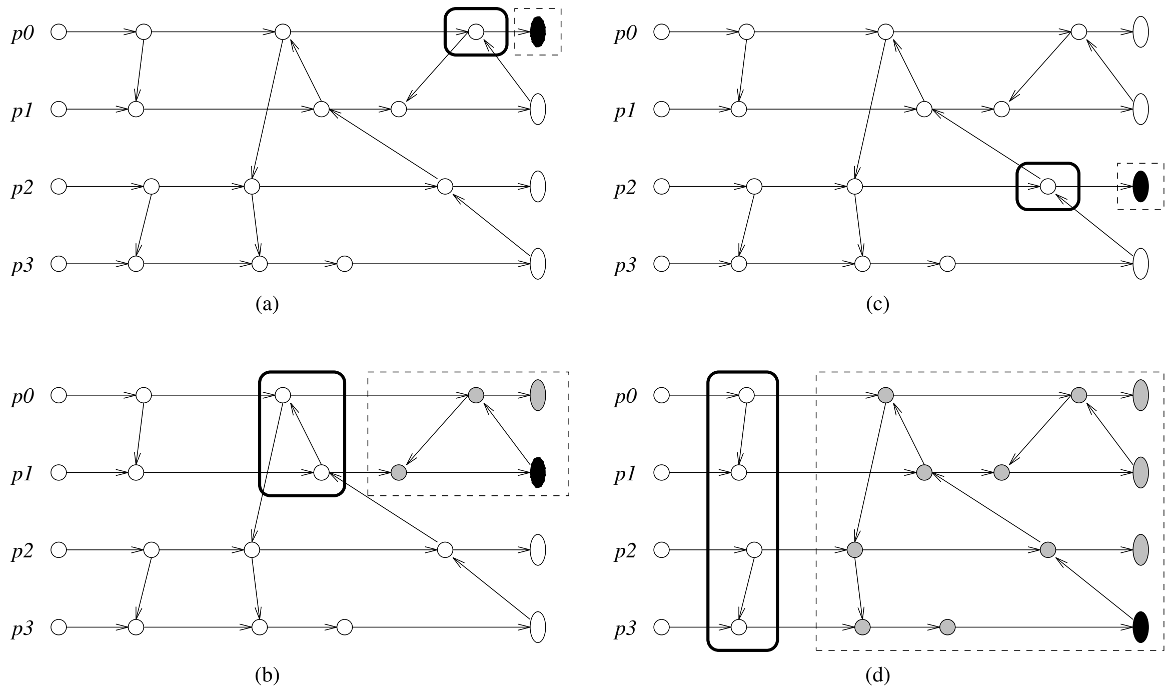

Figure 5: Optimal garbage collection algorithm.

THEOREM 2: tight upper bound

The number of useful checkpoints is bounded by $N(N+1)/2$, where N is the number of processes, and the bound is tight.

Proof. Each of N immediate recovery lines $I(G, v)$, $v \in V$, consists of N checkpoints, one from each process. First, of these N^2 checkpoints, the checkpoint of p_i in $I(G, v_i)$ must be nonvolatile and so N such checkpoints must be useful. Of the remaining $N^2 - N$ checkpoints, we divide them into $(N^2 - N)/2$ pairs where each pair consist of p_i's checkpoint in $I(G, v_j)$ and p_j's checkpoint in $I(G, v_i)$ $(i \neq j)$. If $v_j \not\rightarrow v_i$ and $v_i \not\rightarrow v_j$, then the pair does not contribute any nonvolatile useful checkpoint; if $v_j \rightarrow v_i$ and $v_i \not\rightarrow v_j$ (or $v_i \rightarrow v_j$ and $v_j \not\rightarrow v_i$), then the pair may contribute one additional useful checkpoint; if $v_j \rightarrow v_i$ and $v_i \rightarrow v_j$, then $R(v_i) = R(v_j)$ and $I(G, v_i) = I(G, v_j)$, and the pair are part of the first N useful checkpoints. Hence, each pair contributes at most one additional useful checkpoint. Therefore, the number of useful checkpoints is bounded by $N + ((N^2 - N)/2) \times 1 = N(N+1)/2$.

The bound is tight because, given any N, we can construct a nonvolatile R-graph with the structure shown in

Figure 6 to achieve the bound. ■

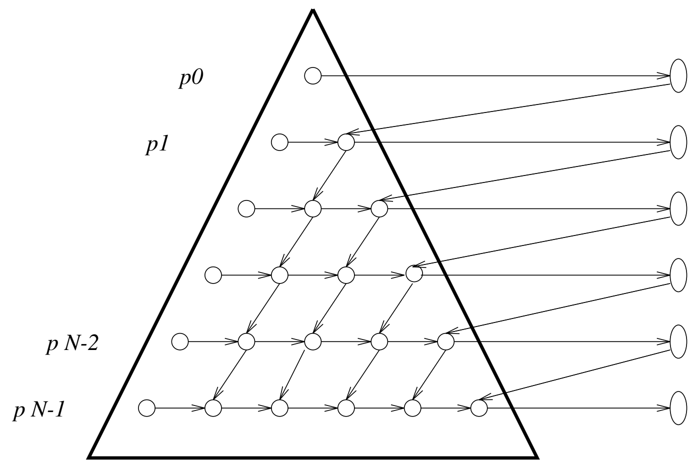

Figure 6: Worst-case R-graph with $N(N+1)/2$ useful checkpoints.

References

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

[2] V. Hadzilacos. An algorithm for minimizing roll back cost. In *Proc. ACM Symp. on Principles of Database Systems*, pages 93–97, 1982.

[3] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[4] B. Randell. System structure for software fault tolerance. *IEEE Trans. Software Eng.*, SE-1(2):220–232, June 1975.

[5] R. E. Strom and S. Yemini. Optimistic recovery in distributed systems. *ACM Trans. Comput. Syst.*, 3(3):204–226, August 1985.

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

[7] Y. M. Wang. *Space Reclamation for Uncoordinated Checkpointing in Message-Passing Systems*. PhD thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, August 1993.

[8] Y. M. Wang. Maximum and minimum consistent global checkpoints and their applications. In *Proc. IEEE Symp. Reliable Distributed Syst.*, pages 86–95, September 1995.

[9] Y. M. Wang, P. Y. Chung, I. J. Lin, and W. K. Fuchs. Checkpoint space reclamation for uncoordinated checkpointing in message-passing systems. *IEEE Trans. Parallel and Distributed Syst.*, 6(5):546–554, May 1995.