Maximum and Minimum Consistent Global Checkpoints and Their Applications

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

Abstract

This paper considers the problem of constructing the maximum and the minimum consistent global checkpoints that contain a target set of checkpoints, and identify it as a generic issue in recovery-related applications. We formulate the problem as a reachability analysis problem on a directed rollback-dependency graph, and develop efficient algorithms to calculate the two consistent global checkpoints for both general nondeterministic executions and piecewise deterministic executions. We also demonstrate that the approach provides a generalization and unifying framework for many existing and potential applications including software error recovery, mobile computing recovery, parallel debugging and output commits.

1 Introduction

A *checkpoint* is a snapshot of the state of a process, saved on nonvolatile storage to survive process failures. It can be reloaded into volatile memory in case of a failure to reduce the amount of lost work. In a message-passing system consisting of N processes, a *global checkpoint* [1] is a set of N local checkpoints, one from each process. A global checkpoint T is *consistent*[1] if no message is sent after a checkpoint of T and received before another checkpoint of T [9].

Most of the literature on checkpointing and rollback recovery considers *hardware failures*. The set of failed processes is uniquely determined by the physical failure, and the most recent consistent global checkpoint, called the *recovery line*, is uniquely determined by the volatile states of surviving processes and the nonvolatile checkpoints [2, 5, 10]. New applications of checkpointing, as described below, pose a different problem: **given a target set of checkpoints, each from a different process, how to construct consistent global checkpoints that contain the set**. For software error recovery, the target set of checkpoints may be suggested by a diagnosis procedure to maximize the chance of bypassing the software bug that caused the error, or it can be a set of checkpoints that were taken after some output actions of which the rollbacks are highly undesirable. In the context of mobile computing, successive checkpoints of a process running on a mobile host may be stored in different locations as the mobile host moves between cells [11], and the target set may consist of available or easily accessible checkpoints. For debugging applications, the target set can be defined by user-specified breakpoints [12].

Given a target set of checkpoints S, the motivation for finding the maximum consistent global checkpoint containing S is often clear: it minimizes the total rollback distance and hence the amount of lost work. The main contribution of this paper is to demonstrate that finding the minimum consistent global checkpoint containing S is an equally important concept that has not been explicitly identified in the literature. In general, it corresponds to the notions of "undo as much as possible during a rollback" or "move forward only if absolutely necessary during a normal execution." For a special class of applications satisfying the MRS model [13] (i.e., all receiving events in a checkpoint interval precede all sending events in the same interval), the concept becomes even more powerful and practically useful because it is tightly coupled with the notion of causality [14] and the technique of transitive dependency tracking [8]. The study of the MRS model is important because it includes the heavily-studied piecewise deterministic model [15] as a special case. We will use several examples to show that the concept of minimum consistent global checkpoints provides a unifying framework for seemingly unrelated applications, and a systematic approach to generalizing exist-

[1] In this paper, we do not address the issue of in-transit messages (also called channel-state messages or lost messages), i.e., messages sent before a global checkpoint and received after. In a low-level implementation, the inconsistency caused by an in-transit message can be taken into account by treating its acknowledge message as a dependency-carrying message and relying on the reliable transmission protocol to retransmit the in-transit message [2–5]. In an application-level implementation, in-transit messages can be retrieved from the receiver logs [6] or the sender logs [7], or regenerated by the senders [8].

1060-9857/95 $04.00 © 1995 IEEE

ing techniques.

This paper is organized as follows. Section 2 defines the rollback-dependency graphs and the maximum and minimum consistent global checkpoints; Section 3 derives the necessary and sufficient condition for a target set of checkpoints to belong to any consistent global checkpoint, and develops two symmetrical algorithms based on reachability analysis of rollback-dependency graphs; Section 4 shows that, for the MRS model, transitive dependency tracking suffices to capture the reachability condition and therefore allows more efficient and distributed implementations of the two algorithms; Section 5 describes four applications of the presented theories and algorithms to demonstrate the capability of our approach to generalize and unify existing techniques.

2 Preliminaries

Let $c_{i,x}$ $(0 \leq i \leq N-1,\ x \geq 0)$ denote the x^{th} checkpoint of process P_i, where i is called the process id and x the *checkpoint index*. Let $I_{i,x}$ $(0 \leq i \leq N-1,\ x \geq 1)$ denote the *checkpoint interval* (or *interval*) between $c_{i,x-1}$ and $c_{i,x}$, as shown in Figure 1(a). Our presentation will be based on a directed **rollback-dependency graph** (or **R-graph**)[2] in which each node represents a checkpoint and a directed edge is drawn from $c_{i,x}$ to $c_{j,y}$ if (1) $i \neq j$, and a message m is sent from $I_{i,x}$ and received in $I_{j,y}$; or (2) $i = j$ and $y = x+1$. (The name "rollback-dependency graph" comes from the observation that if $I_{i,x}$ is rolled back, then $I_{j,y}$ must also be rolled back.) We assume each process P_i starts its execution with an **initial checkpoint** $c_{i,0}$. Clearly, $c_{i,0}$ cannot have any incoming edge and can have only one outgoing edge pointing to $c_{i,1}$.

The edges of an R-graph can be maintained locally by each process using the following direct dependency tracking [1] mechanism:

Direct dependency tracking

When process P_i sends a message m from the interval $I_{i,x}$, the pair (i,x) is piggybacked on m. When the receiver P_j receives m in the interval $I_{j,y}$, the R-graph edge from $c_{i,x}$ to $c_{j,y}$ is recorded by P_j.

The calculation of a consistent global checkpoint can be initiated by any of the N processes or a separate watchdog process [16]. The initiating process broadcasts a *request* message to collect the existing direct dependencies from all the other processes, and constructs the complete R-graph. Each process stops its execution after it replies to the *request* message. The **volatile checkpoint** of each process P_i, which represents the volatile state of P_i when it stops the execution, and its associated direct dependencies are also included in the R-graph. An application-dependent target set of checkpoints S is determined and made available to the initiating process, which then performs reachability analysis on the R-graph to calculate a consistent global checkpoint containing S (as described in the next section). Figure 1(b) shows the R-graph for the example checkpoint and communication pattern in Figure 1(a). To simplify the presentation, we use the same notation $c_{i,x}$ for both volatile and nonvolatile checkpoints in a given checkpoint and communication pattern.

[2] Also called a local system graph in the hardware failure recovery literature [10].

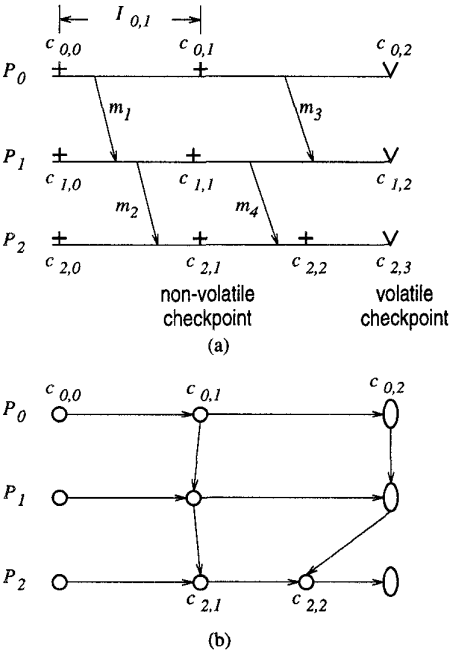

Figure 1: (a) Checkpoint and communication pattern; (b) rollback-dependency graph (R-graph).

Notation

In this paper, the two terms "checkpoint" and "node" will be used interchangeably; a path (or an R-path) always refers to a path in an R-graph. Additional notation is defined below.

- S (possibly with a subscript): a set of checkpoints, each from a different process;
- $S_1 \rightarrow S_2$: there exists a path from a checkpoint in the set S_1 to a checkpoint in the set S_2; if S_1 (or S_2) contains only one checkpoint c, we use $c \rightarrow S_2$ (or $S_1 \rightarrow c$) as a simplified notation for $\{c\} \rightarrow S_2$ (or $S_1 \rightarrow \{c\}$); also, we define $c \rightarrow c$, i.e., every checkpoint is reachable from

itself[3];

• $S_1 \not\rightarrow S_2$: there exists no path from any checkpoint of S_1 to any checkpoint of S_2;

• S_{next}: given a set of checkpoints S, $S_{next} = \{c_{i,x+1} : c_{i,x} \in S \text{ such that } c_{i,x} \text{ is nonvolatile}\}$; note that the size of S_{next} may be smaller than the size of S;

• $T[i]$: given a set of checkpoints T, each from a different process, $T[i]$ denotes the checkpoint from process P_i;

• $index(T[i])$: if $T[i] = c_{i,x}$, then $index(T[i]) = x$.

The definitions of the maximum and the minimum consistent global checkpoints used in this paper are based on the following partially ordered relation.

DEFINITION 1 *Given two consistent global checkpoints T_1 and T_2, $T_1 \preceq T_2$ if and only if $index(T_1[i]) \leq index(T_2[i])$ for all $0 \leq i \leq N-1$.*

DEFINITION 2 *Given a target set of checkpoints, S, each from a different process, let $G(S)$ denote the set of consistent global checkpoints containing S. If $G(S)$ is not empty, then*
(a) $\hat{T} \in G(S)$ is the maximum consistent global checkpoint containing S if and only if $T \preceq \hat{T}$ for all $T \in G(S)$; and
(b) $\check{T} \in G(S)$ is the minimum consistent global checkpoint containing S if and only if $\check{T} \preceq T$ for all $T \in G(S)$.

3 Constructing Maximum and Minimum Consistent Global Checkpoints

The following lemma establishes the fundamental relation between global checkpoint inconsistency and R-graph reachability.

LEMMA 1 A global checkpoint T is inconsistent if and only if $T_{next} \rightarrow T$.

Proof. If $T_{next} \rightarrow T$ By definition there exist $c_{k,u} \in T_{next}$ and $c_{l,v} \in T$ such that $c_{k,u} \rightarrow c_{l,v}$. Consider any path connecting $c_{k,u}$ to $c_{l,v}$. Each node $c_{m,w}$ on the path is either "prior to", "equal to", or "after" $T[m]$. Since the first node $c_{k,u}$ is "after", and the last node $c_{l,v}$ is "equal to", we can find the first node on the path that is not "after". By the construction of R-graphs, the message corresponding to the incoming edge of this node is sent after T and received before T and so T is inconsistent.

If T is inconsistent There must exist a message m sent after some $c_{j,y} \in T$ and received before some $c_{i,x} \in T$. Since P_j cannot send any message after its volatile checkpoint, $c_{j,y}$ cannot be a volatile checkpoint and so $c_{j,y+1}$ must exist. A path from $c_{j,y+1}$ to $c_{i,x}$ can be obtained by proceeding along edges within P_j until the checkpoint immediately following the sending event of m, following the edge corresponding to m, and proceeding along edges within P_i until $c_{i,x}$ is reached. Therefore, $T_{next} \rightarrow T$. ■

[3]Note that, in general, the "$\rightarrow$" relation is not transitive, i.e., $S_1 \rightarrow S_2$ and $S_2 \rightarrow S_3$ do not imply $S_1 \rightarrow S_3$. However, transitivity does apply when S_2 contains only a single checkpoint.

Given a target set of checkpoints S, it may or may not be possible to construct a consistent global checkpoint that contains S. The following theorem demonstrates that the simple reachability test described in Lemma 1 can also be used to determine whether or not it is possible for S to be contained in any consistent global checkpoint. When that is possible, the proof describes two ways of constructing such a global checkpoint, which then lead to two algorithms for constructing the maximum and the minimum consistent global checkpoints containing S, respectively, as proved in Theorem 2.

THEOREM 1 Given a target set of checkpoints S, a consistent global checkpoint containing S exists if and only if $S_{next} \not\rightarrow S$.[4]

Proof. If $S_{next} \rightarrow S$ For any global checkpoint T such that $S \subseteq T$, we must have $S_{next} \subseteq T_{next}$ and so $T_{next} \rightarrow T$; hence, T is inconsistent by Lemma 1.

If $S_{next} \not\rightarrow S$ We describe two ways of constructing a consistent global checkpoint T that contains S.

Construction $\mathcal{A}$

Let T denote the set of **the last checkpoint of each process unreachable from S_{next}**. (The existence of T is guaranteed by the existence of the initial checkpoints, any of which clearly cannot belong to S_{next} and cannot be reachable from S_{next} because of the absence of incoming edges.) The condition $S_{next} \not\rightarrow S$ and the trivial fact that any checkpoint of S_{next} is reachable from S_{next} show that S is contained in T.

If S_{next} is empty, then T consists of all volatile checkpoints and must be consistent because no message is sent after any volatile checkpoint. If S_{next} is not empty then, by construction, $S_{next} \not\rightarrow c$ for all $c \in T$ and $S_{next} \rightarrow c'$ for all $c' \in T_{next}$. We then must have $T_{next} \not\rightarrow T$ because, if there exist $c \in T$ and $c' \in T_{next}$ such that $c' \rightarrow c$, then $S_{next} \rightarrow c' \rightarrow c$ leads to $S_{next} \rightarrow c$, a contradiction. From Lemma 1, T is consistent.

[4]For the special case in which S contains only two checkpoints, Theorem 1 can be viewed as a generalization of Lamport's *happened-before* relation which is a sufficient condition for checkpoint inconsistency, but not necessary [17]. The reachability condition in the statement also provides a uniform view for the notions of zigzag paths and zigzag cycles as presented by Netzer and Xu [17].

Construction $\mathcal{B}$

Let T denote the set of **the last checkpoint of each process that can reach** S. If none of the checkpoints of a process P_i can reach S, then P_i contributes its initial checkpoint $c_{i,0}$ to T. The condition $S_{next} \not\rightarrow S$ and the trivial fact that any checkpoint of S can reach S show that S is contained in T.

By construction, we have $c \rightarrow S$ for any noninitial checkpoint $c \in T$, and $c' \not\rightarrow S$ for all $c' \in T_{next}$. Pick any $c' \in T_{next}$ and suppose there exists $c \in T$ such that $c' \rightarrow c$. Clearly, c must be a noninitial checkpoint and so $c \rightarrow S$. By transitivity, we have $c' \rightarrow S$, a contradiction. So $T_{next} \not\rightarrow T$ and T is consistent by Lemma 1. ■

THEOREM 2 Given a set of checkpoints S such that $S_{next} \not\rightarrow S$, Constructions $\mathcal{A}$ and $\mathcal{B}$ give the maximum and the minimum consistent global checkpoints containing S, respectively.

Proof. Let $\hat{T}$ denote the consistent global checkpoint obtained by Construction $\mathcal{A}$. If S_{next} is empty, then $\hat{T}$ consists of all volatile checkpoints and so must be the maximum; otherwise, for any global checkpoint T containing S such that $\hat{T}[i] < T[i]$ for some $0 \leq i \leq N-1$, we must have $S_{next} \subseteq T_{next}$, $S_{next} \rightarrow T[i]$ (by Construction $\mathcal{A}$), and so $T_{next} \rightarrow T$; hence, T is inconsistent by Lemma 1. We have proved that $T \preceq \hat{T}$ for any consistent global checkpoint T containing S, and so $\hat{T}$ is the maximum.

Let $\check{T}$ denote the consistent global checkpoint obtained by Construction $\mathcal{B}$. For any global checkpoint T containing S such that $T[i] < \check{T}[i]$ for some $0 \leq i \leq N-1, \check{T}[i]$ cannot be an initial checkpoint and so we must have $T_{next} \rightarrow S$ (by Construction $\mathcal{B}$) and so $T_{next} \rightarrow T$; hence, T is inconsistent by Lemma 1. We have proved that $\check{T} \preceq T$ for any consistent global checkpoint T containing S, and so $\check{T}$ is the minimum. ■

The two algorithms that follow Constructions $\mathcal{A}$ and $\mathcal{B}$ are summarized below.

Algorithm $\mathcal{A}$

The algorithm for Construction $\mathcal{A}$ starts a search from every node in S_{next} and marks all reachable nodes. If any node of S is marked during the search, then S cannot belong to any consistent global checkpoint; otherwise, after the search is finished, the last **unmarked** node of each process forms the **maximum** consistent global checkpoint containing S.

Algorithm $\mathcal{B}$

The algorithm for Construction $\mathcal{B}$ starts a search from every node in S, but reversing the direction of all edges, and marks all reachable nodes. If any node of S_{next} is marked during the search, then S cannot belong to any consistent global checkpoint; otherwise, after the search is finished, the last **marked** node of each process is included in the global checkpoint. Processes that do not have any marked nodes contribute their initial checkpoints to complete the **minimal** consistent global checkpoint containing S.

4 Efficient Algorithms for the MRS model

In order to perform the reachability analysis on an R-graph, either a central process needs to collect the edge information from all processes, or the search may be performed in an inefficient, distributed fashion through a potentially large number of messages chasing the edges. One observation is that, if every checkpoint can have on-line knowledge of all the checkpoints that can reach it in the R-graph, then it is possible to develop efficient, distributed algorithms to perform the reachability analysis. One possibility to achieve that is to use *transitive dependency tracking* [7, 8, 11].

Transitive dependency tracking

In a system with N processes, each process P_i maintains a size-N transitive dependency vector D_i. The entry $D_i[i]$ is initialized to 1, incremented every time a new checkpoint (or a logical checkpoint as described later) is encountered, and thus always represents the current interval index or equivalently the checkpoint index of the next checkpoint of P_i; every other entry $D_i[j], j \neq i$, is initialized to -1 and records the highest index of any intervals of P_j on which P_i's current state transitively depends. When P_i sends a message m from the interval $I_{i,x}$, P_i's current D_i vector is piggybacked on m. When the receiver P_j receives m in the interval $I_{j,y}$, P_j updates its D_j vector to be the component-wise maximum of its current D_j and the piggybacked D_i; that is, $D_j[k] = max(D_j[k], D_i[k])$, $0 \leq k \leq N-1$. When P_j reaches its next checkpoint $c_{j,y}$, the vector D_j at that instant is associated with $c_{j,y}$ and denoted by $D_{j,y}$.

In the example shown in Figure 2, the message chain $m_3 - m_2 - m_1$ results in the transitive dependency vector $D_{0,2} = [2, 2, 1, 1]$, and so checkpoint $c_{0,2}$ knows that $c_{1,2}$, $c_{2,1}$ and $c_{3,1}$ are the last checkpoints of P_1, P_2 and P_3, respectively, that can reach $c_{0,2}$. In contrast, the R-path $c_{3,2} \rightarrow c_{2,2} \rightarrow c_{1,2} \rightarrow c_{0,2}$ cannot be on-line tracked. The message m_4 is received by P_1 *after* m_1 was sent, and so the knowledge about the edge from $c_{2,2}$ to $c_{1,2}$ cannot possibly be piggybacked on m_1. It becomes clear that, in order to allow transitive dependency tracking to carry full information about R-graph reachability, we need to impose some constraint on the checkpoint and communication pattern.

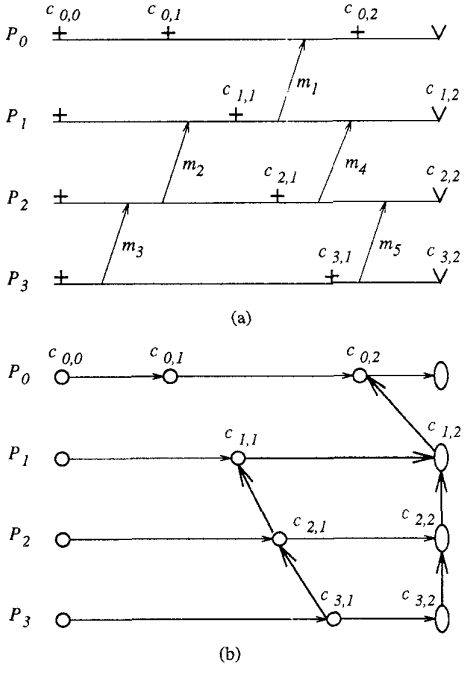

Figure 2: On-line tracking of R-paths. The paths $c_{3,1} \rightarrow c_{0,2}$, $c_{2,1} \rightarrow c_{0,2}$ and $c_{1,2} \rightarrow c_{0,2}$ in (b) can be on-line tracked; but the paths $c_{3,2} \rightarrow c_{0,2}$ and $c_{2,2} \rightarrow c_{0,2}$ cannot be on-line tracked.

4.1 The MRS model

The example in Figure 2 indicates that R-paths are in general not on-line trackable if, within the same checkpoint interval, some message-sending events are allowed to occur before a message-receiving event. Russell [13] introduced the **MRS** model where "M" stands for Mark (or checkpoint), "R" for Receive and "S" for Send. For every process in the MRS model, *all message-receiving events within a checkpoint interval must precede all message-sending events in the same interval.* This can be achieved by taking an additional checkpoint before every message-receiving event that is not separated from its previous message-sending event by a checkpoint. We next show that, for the MRS model, testing the R-graph reachability between any two checkpoints translates into an integer comparison involving entries of transitive dependency vectors. As a result, the maximum and the minimum consistent global checkpoints containing a target set can be computed by more efficient distributed algorithms. In the next section, we will also demonstrate that the study of the MRS model is important because the *piecewise deterministic execution model* [15, 18] that has become quite popular in the recovery literature is in fact a special case of the MRS model. Due to space limitation, the proofs in this subsection are omitted.

LEMMA 2 In the MRS model, for any two checkpoints $c_{i,x}$ and $c_{j,y}$, $c_{j,y} \rightarrow c_{i,x}$ if and only if $D_{i,x}[j] \geq y$.

Based on Lemma 2, we can transform Lemma 1, Theorems 1 and 2 into the following specialized versions for the MRS model.

LEMMA 3 In the MRS model, a global checkpoint T is inconsistent if and only if there exist $c_{i,x}, c_{j,y} \in T$ such that $D_{i,x}[j] > y$.

THEOREM 3 In the MRS model, given a set of checkpoints S, a consistent global checkpoint containing S exists if and only if $D_{i,x}[j] \leq y$ for all $c_{i,x}, c_{j,y} \in S$.

THEOREM 4 In the MRS model, given a set of checkpoints S, the maximum ($\hat{T}$) and the minimum ($\check{T}$) consistent global checkpoints containing S can be computed as follows: for $0 \leq k \leq N-1$,

$$\begin{aligned} (a)\quad & index(\hat{T}[k]) \\ & = \max\{w : D_{k,w}[i] \leq x \ for\ all\ c_{i,x} \in S\}; \\ (b)\quad & index(\check{T}[k]) \\ & = \max(\{0\} \cup \{D_{i,x}[k] : c_{i,x} \in S\}). \end{aligned} \tag{1}$$

The two algorithms for computing the maximum and the minimum consistent global checkpoints in the MRS model are summarized below.

Algorithm $\mathcal{A}'$

Given a target set of checkpoints S in the MRS model, the set of integer pairs (i, x) for every $c_{i,x} \in S$ is sent to every process. Each process P_k can locally determine the k^{th} entry of the **maximum** consistent global checkpoint $\hat{T}$ containing S by searching for the highest-index checkpoint $c_{k,w}$ such that $D_{k,w}[i] \leq x$ for all $c_{i,x} \in S$. If any process P_i with $c_{i,x} \in S$ finds that $index(\hat{T}[i]) \neq x$, then S cannot be contained in any consistent global checkpoint.

Algorithm $\mathcal{B}'$

Given a target set of checkpoints S in the MRS model and their associated transitive dependency vectors $D_{i,x}$ for every $c_{i,x} \in S$, the **minimum** consistent global checkpoint $\check{T}$ containing S can be computed as $index(\check{T}[k]) = \max(\{0\} \cup \{D_{i,x}[k] : c_{i,x} \in S\})$, $0 \leq k \leq N-1$. If the algorithm finds that $index(\check{T}[i]) \neq x$ for any i such that $c_{i,x} \in S$, then S cannot be contained in any consistent global checkpoint.

We note that, while Algorithm $\mathcal{A}'$ in general requires every process to participate by scanning through its dependency vectors, Algorithm $\mathcal{B}'$ can be executed by the algorithm initiator alone if the dependency vectors associated with the checkpoints of S are also available to the initiator. This condition can be easily satisfied when S contains a single checkpoint $c_{i,x}$ and process P_i is the initiator. In this special case, we have the following corollary of Theorem 4(b); Algorithm $\mathcal{B}'$ also reduces to the following Algorithm $\mathcal{B}''$, which has been used in several previous works as described in the next section.

COROLLARY 1 In the MRS model, given a single checkpoint $c_{i,x}$ and its associated transitive dependency vector $D_{i,x}$, the minimum consistent global checkpoint $\check{T}$ containing $c_{i,x}$ can be computed as

Algorithm $\mathcal{B}''$

$$index(\check{T}[k]) = \max(\{0, D_{i,x}[k]\}), 0 \leq k \leq N-1 \quad (2)$$

We would like to point out that Eq. (2) bears a similarity to the well-known fact that the vector timestamp of event e captures exactly the set of all events which causally precede e [14]. The main difference is that we may not have a checkpoint for every process state at any point in time; hence, the problem of finding "consistent global checkpoints" is in general different from that of determining "consistent global states." We will next show that, under piecewise determinism, we can have a logical checkpoint for every process state and so the two problems become identical. The example of causal distributed breakpoints in the next section further demonstrates this point.

4.2 Piecewise deterministic model: A special case of MRS

We next show that applications satisfying the assumption of piecewise determinism (PWD) must satisfy the MRS model, and so transitive dependency vectors suffice to determine checkpoint consistency. Under the PWD assumption, each process execution is divided into deterministic *state intervals* $S_{i,x}$ bounded by message-receiving events or other nondeterministic events, as illustrated in Figure 3(a). It is assumed that every nondeterministic event can be identified, recorded (or logged), and replayed. As a result, every process state can be recreated by replaying the event logs in their original order, even though the state is not directly checkpointed. For example, process P_i in Figure 3 can recreate the state interval $S_{i,x+1}$ by restoring the *physical checkpoint* C, and replaying m_x, e and m_y in that order. Equivalently, it can be modeled as having an additional *logical checkpoint* [19] anywhere inside $S_{i,x+1}$. For recovery applications, it usually suffices to place a logical checkpoint at the *end* of each state interval, as shown in Figure 3(b). (For some debugging applications, it may be desirable to place an additional logical checkpoint immediately after every message-sending event to allow replaying only part of a state interval, as described in the next section.) Under the logical checkpoint model, any application satisfying piecewise determinism must have a logical checkpoint before every message-receiving event, and therefore must satisfy the MRS model. A logical checkpoint is called *nonvolatile* (or stable) if all event logs since the previous physical checkpoint have been saved onto stable storage; otherwise, it is a *volatile logical checkpoint.*

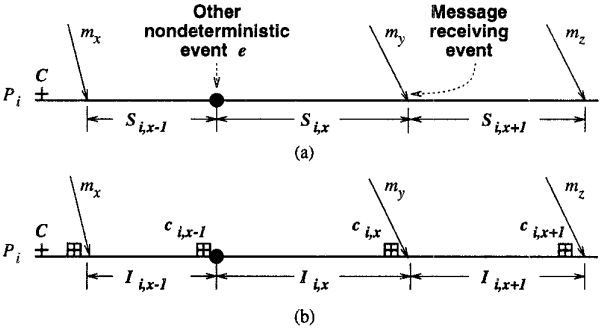

Figure 3: (a) Piecewise deterministic execution model consisting of state intervals; (b) the corresponding MRS model consisting of logical checkpoint intervals.

5 Applications

5.1 Software error recovery

Distributed service-providing applications typically have periodic coordinated checkpoints to bound the rollback distance, and take additional uncoordinated checkpoints[5] to further localize the recovery [20, 21]. Studies have shown that, since most software errors in production software are transient, rollback retry can often provide an effective way of bypassing software bugs [19, 22, 23]. Suppose a software error is detected at the point marked "X" in Figure 4(a), possibly caused by an unexpected nondeterministic event. A diagnosis procedure examines the error symptom and determines that the system should roll back to a state containing checkpoints C and D to maximize the chance of recovery. To minimize the rollback distance, Algorithm $\mathcal{A}$ can be used to find the **maximum**

[5]Possibly in the form of logical checkpoints if message logging/replay is employed for the parts of process execution that satisfy the piecewise deterministic assumption.

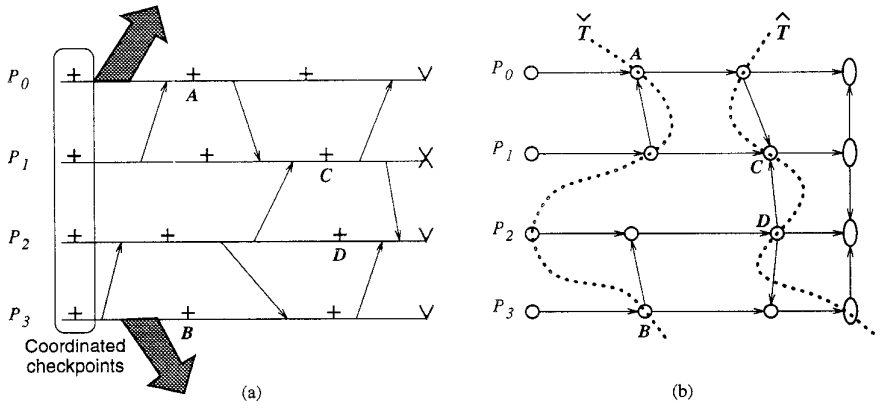

Figure 4: (a) Checkpoint and communication pattern with outputs (shaded arrows); (b) R-graph ($\check{T}$ is the minimum consistent global checkpoint containing $\{A,B\}$, and $\hat{T}$ is the maximum consistent global checkpoint containing $\{C,D\}$).

consistent global checkpoint $\hat{T}$ containing C and D, as shown in Figure 4(b).

If the above localized retry (which does not roll back P_3) still leads to the same error, then a more expensive global rollback to the most recent coordinated checkpoints should be initiated. However, if the rollback will involve revoking some outputs sent to the clients, the associated penalties may suggest that a retry that does not revoke any output be performed first. For example, in Figure 4(a), the system may wish to roll back as far as possible, i.e., as close to the coordinated checkpoints as possible, in order to allow more nondeterminism to bypass the bug; but at the same time, the system does not want to roll back past the two output points indicated by shaded arrows. The problem translates into finding the **minimum** consistent global checkpoint containing checkpoints A and B, and Algorithm $\mathcal{B}$ can be used to obtain the global checkpoint $\check{T}$, as shown in Figure 4(b).

5.2 Mobile computing recovery

Acharya and Badrinath [11] described an algorithm for collecting consistent global checkpoints in distributed applications running on mobile computers. We will show that their algorithm basically computes the **minimum** consistent global checkpoint containing a given checkpoint. The intention is not to determine the earliest consistent global checkpoint, but to minimize the searching cost for locating the needed checkpoints by taking advantage of the simplicity and the on-line capability of Algorithm $\mathcal{B}''$ for the MRS model.

A typical architecture for supporting mobile computing [24] is illustrated in Figure 5. The entire geographic area is partitioned into wireless *cells* and each cell is covered by a Mobile Support Station (MSS) which provides wireless interface to communicate with *mobile hosts* (MH) in the cell. This architecture introduces three new issues for collecting consistent global checkpoints: first, the nonvolatile storage attached to mobile hosts cannot be considered as stable storage; second, mobile hosts may move from one cell to another; third, a mobile host may voluntarily disconnect from the rest of the network.

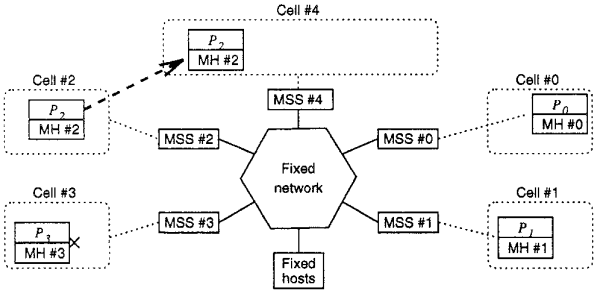

Figure 5: Typical architecture for mobile computing with one Mobile Support Station (MSS) supporting all Mobile Hosts (MH's) in each cell.

To address the first issue, the authors proposed that the checkpoints of a process running on a mobile host should be stored in its local MSS. Consequently, as a mobile host moves between cells, its successive checkpoints may be stored on different MSS's. To facilitate locating the checkpoints, each process P_i maintains a *location vector* L_i in which $L_i[j]$ records the cell id of the MSS that stores checkpoint $c_{j,D_i[j]}$. The vector L_i is piggybacked on each outgoing message m, and the receiver P_k of m sets $L_k[j] = L_i[j]$ when it sets $D_k[j] = D_i[j]$ in the component-wise maximum operation.

Let $L_{i,x}$ denote the location vector of P_i when check-

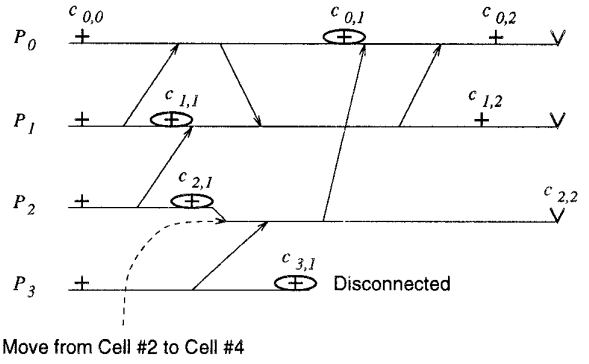

Figure 6: Checkpoint and communication pattern with process P_2 moving between cells and P_3 disconnected.

point $c_{i,x}$ is taken. Figure 6 gives an example where P_2 moves between cells and P_3 voluntarily disconnects. Additional checkpoints need to be taken under three conditions: (1) checkpoints $c_{0,1}$ and $c_{1,1}$ are taken to maintain the MRS structure; (2) $c_{2,1}$ is taken immediately before MH #2 moves from Cell #2 to Cell #4 to guarantee each interval corresponds to one location; (3) $c_{3,1}$ is taken immediately before MH #3 voluntarily disconnects because P_3 may not be active to provide its volatile checkpoint as part of the consistent global checkpoint being collected. Suppose P_0 wants to collect a consistent global checkpoint containing $c_{0,2}$ without incurring a large search overhead. It can use Algorithm $\mathcal{B}''$ to compute the minimum consistent global checkpoint as $\check{T} = \{c_{0,2}, c_{1,2}, c_{2,2}, c_{3,1}\}$ based on $D_{0,2}$, and locate the checkpoints based on $L_{0,2} = \{0, 1, 4, 3\}$. Our derivation in Section 4 shows that the above scheme can be easily generalized by using Algorithm $\mathcal{B}'$, instead of Algorithm $\mathcal{B}''$, to compute the minimum consistent global checkpoint containing more than one checkpoint, for example, multiple checkpoints stored in the same MSS. This will in general obtain a "more recent" consistent global checkpoint (comparing Eq. (1) with Eq. (2)), while still allowing the constituent checkpoints to be easily located.

5.3 Debugging under piecewise determinism

Figure 7(a) shows a checkpoint and communication pattern with logical checkpoints under the PWD assumption. Suppose A is a breakpoint specified by a debugger user, and the objective is to find a consistent distributed breakpoint that contains A. The minimum consistent global checkpoint containing A, which is computed as $\{A, B, C, D\}$ by Algorithm $\mathcal{B}''$, is one possible choice. However, Fowler and Zwaenepoel [12] argued that a distributed breakpoint containing A should reflect all the interesting states that causally affect A. For example, the state of P_1 at the point "X", which immediately follows the sending event of m_3, may provide useful information as to why m_3 is generated to causally affect A. Such information may be destroyed if P_1's state is reconstructed up to B. They then define a *causal distributed breakpoint* which is basically the **minimum** consistent global checkpoint containing a given breakpoint with an additional logical checkpoint (conceptually) being added to immediately follow every message-sending event, as illustrated in Figure 7(b). Since it still satisfies the MRS model, Algorithm $\mathcal{B}''$ can be used to determine the causal distributed breakpoint containing A to be $\{A, X, Y, Z\}$. Alternatively, direct dependency tracking can be used to reduce message piggybacking overhead, in which case either a depth-first search [12] or a breadth-first search [14] needs to be performed in a distributed fashion to iteratively construct the transitive dependency vector and hence the causal distributed breakpoint. Again, we can generalize the definition of causal distributed breakpoints to allow multiple user-specified breakpoints, and use Algorithm $\mathcal{B}'$ to compute the generalized causal distributed breakpoints.

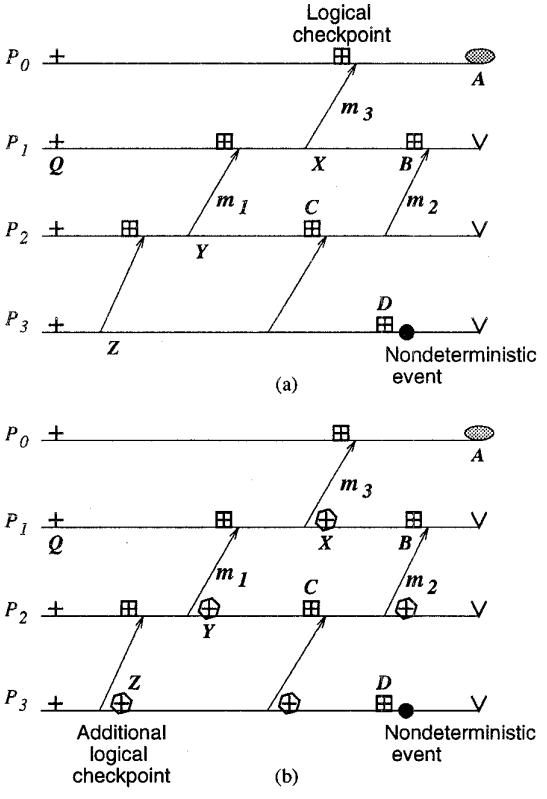

Figure 7: Piecewise deterministic model (a) for recovery; (b) for causal distributed breakpoints.

5.4 Output commit

The issue of output commits has mostly been studied in the literature on hardware failure recovery under piecewise determinism [7]. When an application sends an output to the "outside world" which is not capable of performing rollbacks, it must ensure that the state interval S from which the output is sent must be recreatable so that under no circumstances will it revoke the output. An output commit under piecewise determinism generally involves making all the state intervals, that S depends on, nonvolatile and hence recreatable after a failure. An alternative interpretation of such an output commit action is to find a consistent global checkpoint T containing the logical checkpoint A at the end of S, and make T nonvolatile. This guarantees that any future recovery will not roll back past T and so the output will never be revoked. Since making an earlier logical checkpoint nonvolatile generally requires fewer messages to be saved onto stable storage, the **minimum** consistent global checkpoint containing A is the best choice to minimize the delay before the output can be committed.

The alternative interpretation has two advantages. First, it can be readily applied to applications that contain a mix of PWD and non-PWD processes [25]. Figure 8(a) gives an example in which processes P_0 and P_1 are PWD processes, P_2 is a non-PWD process, and P_3 starts as a PWD process but "turns off" PWD after a nonreplayable nondeterministic event e is encountered. The shaded arrow represents an output commit, and so the required action is to make the minimum consistent global checkpoint $\check{T}$ containing A become nonvolatile. Figure 8(b) shows the corresponding R-graph, and $\check{T} = \{A, B, C, D\}$ is determined by Algorithm $\mathcal{B}$. Logical checkpoints A and B can be made nonvolatile by logging all the nondeterministic events since the physical checkpoints Q and R, respectively, onto stable storage. Volatile checkpoints C and D can only be made nonvolatile by saving their states onto stable storage as physical checkpoints. The second advantage of the alternative interpretation is that it demonstrates that the problem of output commits and the problem of "minimal coordination," i.e., involving the minimum number of processes in coordinated checkpointing [3, 26], are in fact the same problem. Suppose process P_0 initiates a coordinated checkpointing session by taking a local checkpoint A. By discarding all obsolete checkpoints and treating the latest coordinated checkpoints as the initial checkpoints, the minimal coordination problem then translates into finding the minimum consistent global checkpoint $\check{T}$ containing A, and making $\check{T}$ nonvolatile.

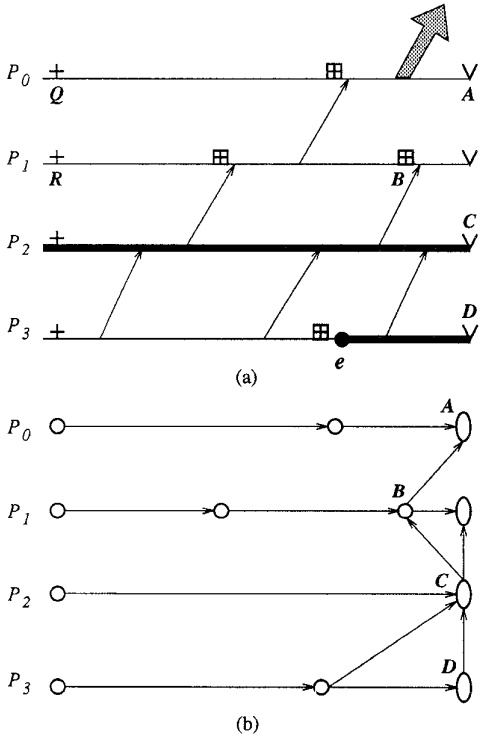

Figure 8: Output commit (shaded arrow) in a system with a mix of piecewise deterministic (PWD) and non-PWD processes. (Thick lines indicate non-PWD executions.)

6 Summary

This paper has extended traditional recovery-related problems in two dimensions. First, the recovery lines may not always be uniquely determined by physical hardware failures; they can be specified to contain a target set of checkpoints for other recovery purposes such as software error recovery. Second, while the maximum consistent global checkpoint appears to be the most natural concept for recovery, the minimum consistent global checkpoint is an equally important concept as it provides a uniform view for many recovery and debugging issues.

For any future application which can be modeled as finding either the maximum or the minimum consistent global checkpoint containing a target set of checkpoints in either a general or an MRS model, the algorithms and theorems presented in this paper can be directly applied to avoid the needs of application-specific correctness proofs, and to serve as the basis for further optimizations.

Acknowledgement

The author wishes to express his sincere thanks to David Johnson and Jian Xu for their valuable discussions, and to Gaurav Suri, Robert Netzer, Emerald Chung, Arup Acharya, and the anonymous referees for their useful comments.

References

[1] Y. M. Wang, A. Lowry, and W. K. Fuchs, "Consistent global checkpoints based on direct dependency tracking," *Inform. Process. Lett.*, Vol. 50, No. 4, pp. 223–230, May 1994.

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

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

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

[5] Y. M. Wang, "Space reclamation for uncoordinated checkpointing in message-passing systems." Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Aug. 1993.

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

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

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

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

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

[11] A. Acharya and B. R. Badrinath, "Checkpointing distributed applications on mobile computers," in *Proc. the Third International Conference on Parallel and Distributed Information Systems*, Sept. 1994.

[12] J. Fowler and W. Zwaenepoel, "Causal distributed breakpoints," in *Proc. IEEE Int. Conf. Distributed Comput. Syst.*, pp. 134–141, 1990.

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

[14] R. Schwarz and F. Mattern, "Detecting causal relationships in distributed computations: in search of the holy grail," *Distributed Computing*, Vol. 7, pp. 149–174, 1994.

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

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

[17] R. H. B. Netzer and J. Xu, "Necessary and sufficient conditions for consistent global snapshots," *IEEE Trans. Parallel and Distributed Syst.*, Vol. 6, No. 2, pp. 165–169, Feb. 1995.

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

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

[20] G. Suri, Y. Huang, Y. M. Wang, W. K. Fuchs, and C. Kintala, "An implementation and performance measurement of the progressive retry technique," in *Proc. IEEE International Computer Performance and Dependability Symposium*, pp. 41–48, Apr. 1995.

[21] Y. Huang and Y. M. Wang, "Why optimistic message logging has not been used in telecommunications systems," in *Proc. IEEE Fault-Tolerant Computing Symp.*, 1995.

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

[23] I. Lee and R. K. Iyer, "Faults, symptoms, and software fault tolerance in the Tandem GUARDIAN90 operating system," in *Proc. IEEE Fault-Tolerant Computing Symp.*, 1993.

[24] T. Imielinski and B. R. Badrinath, "Mobile wireless computing," *Commun. ACM*, Vol. 37, No. 10, pp. 18–28, Oct. 1994.

[25] D. B. Johnson, "Efficient transparent optimistic rollback recovery for distributed application programs," in *Proc. IEEE Symp. Reliable Distributed Syst.*, pp. 86–95, 1993.

[26] P. Leu and B. Bhargava, "Concurrent robust checkpointing and recovery in distributed systems," in *Proc. Int'l Conf. on Data Engineering*, Feb. 1988.