Necessary and Sufficient Conditions for Consistent Global Snapshots

Robert H. B. Netzer and Jian Xu

Abstract—**Consistent global snapshots are important in many distributed applications. We prove the exact conditions for an arbitrary checkpoint, or a set of checkpoints, to belong to a consistent global snapshot, a previously open problem. To describe the conditions, we introduce a generalization of Lamport's happened-before relation called a *zigzag path*.**

Index Terms— **Causality, global checkpoints, distributed systems, consistent global states, Lamport's happened-before relation.**

I. Introduction

CHECKPOINTS are used in many applications including parallel debugging [6], [10], distributed simulation [3], and fault-tolerant computing [1], [4]. A checkpoint is a saved intermediate state of an execution and can be analyzed post-mortem or used to restart the execution from that point. A global snapshot[1] of a distributed computation is a set of local checkpoints, one for each process. A global snapshot is *consistent* if no local checkpoint *happens before* another; that is, there is no causal path from one checkpoint to another in the sense that a message (or sequence of messages) sent after one checkpoint is received before the other [5]. As a result, a consistent snapshot is a set of local states that actually occurred simultaneously during execution or had the potential of occurring simultaneously. Determining when individual local checkpoints can be combined with others to form a consistent snapshot is an important problem. In this paper we prove the exact conditions under which an arbitrary checkpoint, or a set of checkpoints, can be combined with other checkpoints to form a consistent global snapshot. In the past only necessary conditions have been known; necessary and sufficient conditions have been an open problem.

Our main result is a proof of the necessary and sufficient conditions. For example, given two checkpoints, the definition of consistency says that neither can happen before the other if they are to belong to the same consistent snapshot; this condition is necessary for the checkpoints to be consistent. In Fig. 1, neither of the checkpoints $C_{1,1}$ and $C_{3,1}$ happen before each other, and they can be combined with $C_{2,1}$ to form a consistent snapshot (as shown by the dashed line). However, as noticed by Wang, Lowry, and Fuchs [8], this condition is not sufficient. Checkpoints $C_{1,1}$ and $C_{3,2}$ also do not happen before each other, but they *cannot* be combined with any other checkpoint that was taken during execution to form a consistent snapshot. A consistent snapshot contains one checkpoint per process, and in this example no checkpoint in $p2$ was taken that can be combined with both $C_{1,1}$ and $C_{3,2}$ while maintaining consistency. Because of message $m4, C_{3,2}$ cannot be consistent with $C_{2,1}$ or any earlier checkpoint in $p2$, and because of message $m3, C_{1,1}$ cannot be consistent with $C_{2,2}$ or any later checkpoint in $p2$. Thus, no checkpoint in $p2$ is available to form a consistent snapshot.

To describe the necessary and sufficient conditions, we define a generalization of Lamport's happened-before relation called a *zigzag path*. The happened-before relation shows causal paths among checkpoints; one checkpoint happens before another if a sequence of messages exists from one checkpoint to another with each message sent after the previous one in the sequence is received. A zigzag path between two checkpoints is like a causal path, but a zigzag path allows a message to be sent *before* the previous one in the path is received. As we later prove, a zigzag path exists between two checkpoints exactly when they cannot belong to the same consistent snapshot. For example, in Fig. 1 the checkpoint $C_{1,0}$ happens before $C_{3,1}$ because of the causal path formed by messages $m1$ and $m2$. Although a causal path does not exist from $C_{1,1}$ to $C_{3,2}$, a zigzag path does exist, formed by messages $m3$ and $m4$ (the path forms a zigzag shape in the graph). This zigzag path means that no consistent snapshot can be formed (from the execution's checkpoints) that contains *both* $C_{1,1}$ and $C_{3,2}$.

Zigzag paths have applications to any problem that requires saving or analyzing consistent snapshots. For example, in a previous paper [11] we used zigzag paths to virtually eliminate rollback propagation in a checkpointing and rollback recovery scheme for fault tolerance [2]. In such a scheme, the state of a distributed system is saved periodically in checkpoints during normal execution. Upon detecting a fault, recovery is performed by rolling the system back to a previous state not affected by the fault, and resuming normal execution. The snapshot from which restart occurs must be consistent to ensure no process is restarted from a state that has recorded the receipt of a rolled back message. In an independent checkpointing scheme (where there is no checkpoint coordination among processes), not all checkpoints are guaranteed to belong to a consistent snapshot, and in the worst case, rollback propagates and requires restarting from the execution's beginning [7]. We virtually eliminate rollback propagation by tracking zigzag paths on-line, allowing each process to adaptively take checkpoints at moments that tend to minimize the number of checkpoints that cannot belong to a consistent snapshot [11].

Manuscript received July 1993; revised December 1993. This research was partly supported by ONR Contract N00014-91-J-4052, ARPA Order 8225.

The authors are with the Department of Computer Science, Brown University, Providence, RI 02912 USA (e-mail: rn@cs.brown.edu).

IEEE Log Number 9408133.

[1] The terms *global snapshot* and *global checkpoint* are used interchangeably in the literature. We use the term "global snapshot" to avoid overloading the term "checkpoint" with two meanings (one global and one local).

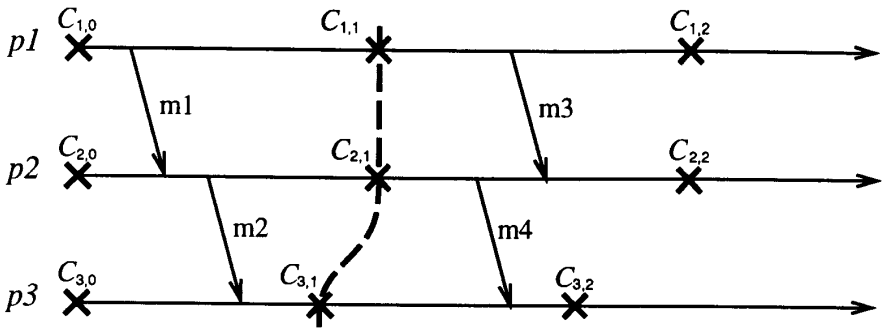


Fig. 1. Consistent and inconsistent snapshots ("×"s indicate checkpoints): $C_{1,1}, C_{2,1}$ and $C_{3,1}$ form a consistent snapshot; $C_{1,1}$ and $C_{3,2}$ cannot belong together in a consistent snapshot.

Another application for consistent snapshots and consistent states is parallel and distributed debugging. A consistent state represents some state that could have occurred during an execution regardless of the relative speed of process execution or message delays. Only consistent states can be used to evaluate certain global predicates or answer debugging queries. We believe the work presented here also provides a better understanding of consistent global states.

II. Model

We next outline our model for representing executions of message-passing programs. A *program execution* is a pair, $P = \langle E, \xrightarrow{\text{HB}} \rangle$, where E is a finite set of *events* and $\xrightarrow{\text{HB}}$ is the *happened-before* relation defined over E [5]. An event represents the execution instance of a send, receive, or checkpoint operation.

We assume that a fixed number of processes exist during execution, and that each process periodically saves (or *checkpoints*) its state in stable storage. We denote the ith checkpoint in process p as $C_{p,i}$ (but sometimes denote checkpoints as simply A, B, or C for clarity). We also assume each process p takes an initial checkpoint $C_{p,0}$ immediately before execution begins, and ends with a *virtual* checkpoint that represents the last state attained before termination [1], [9]. The ith *checkpoint interval* of a process p is all the computation performed between its ith and $i+1$st checkpoints (and includes the ith checkpoint but not the $i+1$st).

The happened-before relation, $\xrightarrow{\text{HB}}$, shows how events potentially affect one another [5], and is defined as the irreflexive transitive closure of the union of two other relations: $\xrightarrow{\text{HB}} = (\xrightarrow{\text{XO}} \cup \xrightarrow{\text{M}})^+$. The $\xrightarrow{\text{XO}}$ relation shows the order in which events in the same process execute. The ith event in any process p (denoted $e_{p,i}$) always executes before the $i+1$st event: $e_{p,i} \xrightarrow{\text{XO}} e_{p,i+1}$. The $\xrightarrow{\text{M}}$ relation shows the order in which messages are delivered: $a \xrightarrow{\text{M}} b$ means that a sent a message that b received (we also write $a \xrightarrow{\text{M}} b$ to denote the message a sent). An event a is said to *happen before* (or have a *causal path* to) an event b iff a could affect b because they belong to the same process or because a sequence of messages was sent from a (or a following event) to b (or a preceding event). In addition, we assume that communication channels are reliable and deliver messages in *FIFO* order.

A *global snapshot* is a set of local checkpoints, one per process (for simplicity, we assume the channel states can be inferred from the local states). A *consistent global snapshot* is a global snapshot in which no message is recorded as received but not yet sent. That is, any two checkpoints A and B in a consistent global snapshot are *unordered* by the $\xrightarrow{\text{HB}}$ relation (i.e., $\neg(A \xrightarrow{\text{HB}} B) \wedge \neg(B \xrightarrow{\text{HB}} A)$).

III. Zigzag Paths and Consistent Global Snapshots

We now present our results. We first introduce *zigzag paths* as a generalization of Lamport's happened-before relation, and then use them to characterize exactly when an arbitrary set of checkpoints can all belong to the same consistent snapshot. We then present examples of two important special cases: the conditions for an arbitrary checkpoint to be useful in the sense that some consistent snapshot exists to which it belongs, and the conditions for two arbitrary checkpoints to both belong to the same consistent snapshot.

A. Zigzag Paths

Recall that a global snapshot is consistent iff none of its component checkpoints happen before one another (they are all mutually unordered). However, as shown in Fig. 1, having two checkpoints that do not happen before each other is not sufficient for ensuring that they can belong together in some consistent snapshot. Even if neither happens before the other, they can be precluded from ever belonging to a consistent snapshot by their relationships to checkpoints in other processes. We define the notion of a *zigzag path* as a generalization of Lamport's happened-before relation to express this situation.

Definition 1: A *zigzag path* exists from $C_{p,i}$ to $C_{q,j}$ iff there are messages $m_1, m_2, \cdots, m_n$ $(n \geq 1)$ such that

1) m_1 is sent by process p after $C_{p,i}$,
2) if m_k $(1 \leq k < n)$ is received by process r, then m_{k+1} is sent by r in the same or a later checkpoint interval (although m_{k+1} may be sent before *or* after m_k is received), and
3) m_n is received by process q before $C_{q,j}$.

Checkpoint C is involved in a *zigzag cycle* iff there is a zigzag path from C to itself.

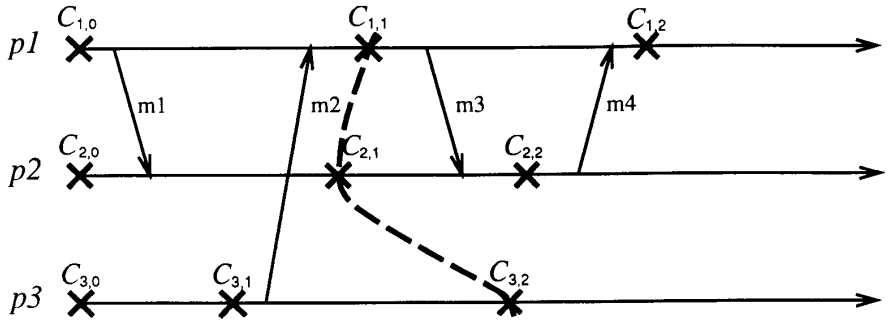


Fig. 2. A zigzag cycle from $C_{2,2}$ back to itself and a consistent snapshot including $C_{1,1}$ and $C_{3,2}$ (between which no zigzag path exists).

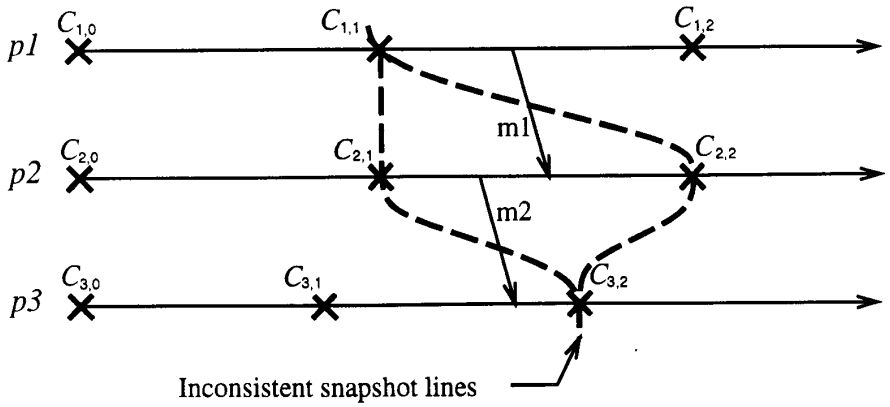


Fig. 3. The zigzag path from $C_{1,1}$ to $C_{3,2}$ renders any snapshot line through $C_{1,1}$ and $C_{3,2}$ inconsistent. $m1$ and $m2$ order the two checkpoints connected by the line they cross, making either snapshot inconsistent.

It is important to note the difference between a zigzag path and a causal path. Lamport's $\xrightarrow{\text{HB}}$ relation shows causal paths. A causal path exists from A to B iff there is a chain of messages starting after A and ending before B with each message sent after the previous one in the chain is received. For a zigzag path, we allow such a chain, but also allow any message in the chain to be sent *before* the previous one is received, as long as the send and receive are in the same checkpoint interval. Thus, a causal path is always a zigzag path, but a zigzag path may not be a causal path (and thus may not cause one checkpoint to happen before the other). Fig. 1 illustrates such a difference. A causal path exists from $C_{1,0}$ to $C_{3,1}$ formed by the chain of messages $m1$ and $m2$, the receive of $m1$ happening *before* the send of $m2$; this causal path is also a zigzag path. A zigzag path exists from $C_{1,1}$ to $C_{3,2}$ formed by the chain of messages $m3$ and $m4$, the receive of $m3$ happening *after* the send of $m4$; this zigzag path is not a causal path, since it does not cause $C_{1,1}$ to happen before $C_{3,2}$.

Another difference between zigzag paths and causal paths is that zigzag paths do not always represent causality, so it is possible for a zigzag path to exist from a checkpoint back to itself (called a zigzag cycle). In contrast, causal paths can never form cycles. A zigzag cycle can exist because in a zigzag path we allow a message to be sent before the previous message in the path is received (as long as the send and receive are in the same checkpoint interval). Fig. 2 shows a zigzag cycle involving $C_{2,2}$, formed by messages $m4$ and $m3$; $m3$ is sent before $m4$ is received, and both occur in the same checkpoint interval.

B. Consistent Global Snapshots

Our definition of zigzag paths generalizes the happened-before relation since causal paths are always zigzag paths but not vice versa. Making such a generalization allows us to capture exactly what constrains checkpoints to be in the same consistent snapshot. In this section, we prove that a consistent snapshot can be formed that includes a set S of checkpoints iff no zigzag path exists between any two checkpoints in S (including a zigzag cycle from any checkpoint to itself).

Intuitively, if a zigzag path exists between a pair of checkpoints, and that zigzag path is also a causal path, then the checkpoints are ordered. Thus, they cannot be consistent, because consistency requires checkpoints to be unordered. However, even if the zigzag path is not a causal path, a consistent snapshot cannot be formed that contains both checkpoints. The zigzag nature of the path causes any snapshot that includes the two checkpoints to be inconsistent. To visualize the effect of a zigzag path, consider any *snapshot line* (a line drawn through a set of checkpoints) through the two checkpoints. Because of the zigzag path between the checkpoints, the line will always cross a message that causes one of the checkpoints to happen before the other, meaning that the snapshot is inconsistent. Fig. 3 shows two snapshot lines through $C_{1,1}$

and $C_{3,2}$ (shown by the dashed lines). The zigzag path from $C_{1,1}$ to $C_{3,2}$ renders both of these snapshot lines inconsistent, since messages $m1$ and $m2$ cross either snapshot line in way that orders two of its checkpoints.

Conversely, if no zigzag path exists between two checkpoints (including a zigzag path from a checkpoint to itself), then it is *always* possible to construct a consistent snapshot containing them. By including the first checkpoint in each process that has no zigzag path to either checkpoint, a consistent snapshot is formed. The snapshot is guaranteed to be consistent since none of its component checkpoints can happen before one another (otherwise, at least one of its checkpoints has a zigzag path to one of the two checkpoints). It is important to note that messages can cross a consistent snapshot line, as long as they do not cause any of the line's checkpoints to happen before each other. For example, $C_{1,2}$ and $C_{2,2}$ can be combined with $C_{3,1}$ to form a consistent snapshot; even though $m2$ would then cross the snapshot line, it would not order any of the checkpoints on the line. The following theorem proves these observations.

Theorem 1 (Consistency Theorem): A set of checkpoints S, where each is from a different process, can belong to the same consistent global snapshot iff no checkpoint in S has a zigzag path to any other checkpoint (including itself) in S.

Proof: If Part: We will show that the absence of the zigzag paths implies that checkpoints in S can belong to the same consistent snapshot. We construct a consistent global snapshot G that contains the checkpoints in S plus one checkpoint in each other process as follows:

1) For each process that has no checkpoint in S and that has a checkpoint with a zigzag path to a member of S, we include in G its *first* checkpoint that has no zigzag path to any checkpoint in S. Such a checkpoint is guaranteed to exist because each process' virtual checkpoint (the checkpoint representing the last state it attained before terminating) never has an outgoing zigzag path.
2) For each process that has no checkpoint in S and that has no checkpoint with a zigzag path to a member of S, we include its initial checkpoint in G. Note that no other checkpoint can happen before this initial checkpoint.

We claim that all checkpoints in G are mutually unordered (so G is consistent). To establish a contradiction, assume that one checkpoint in G happens before another.

Case 1: $A \in S$ happens before $B \in S$. This condition implies that a zigzag path also exists from A to B, contradicting the assumption that no zigzag path exists between any two checkpoints in S.

Case 2: $A \in G - S$ happens before $B \in S$. This condition implies that a zigzag path also exists from A to B, contradicting the way $G - S$ is constructed (by conditions (1) and (2) above). The checkpoints in $G - S$ are chosen in such a way that they have no zigzag paths to any member of S.

Case 3: $A \in S$ happens before $B \in G - S$. B cannot be an initial checkpoint, since no checkpoint can happen before an initial checkpoint. Thus, B must be the first checkpoint in its process that has *no* zigzag path to a member of S (condition (1) above). The checkpoint preceding B must have a zigzag path to a member of S, say checkpoint C (otherwise, the checkpoint preceding B would have been included in G instead of B). This zigzag path, plus the messages that cause A to happen before B, establish a zigzag path from A to C, contradicting the assumption.

Case 4: $A \in G - S$ happens before $B \in G - S$. By the argument in case 3, a zigzag path must exist from A to some checkpoint in S. But this contradicts the assumption that A is the first checkpoint in its process with no zigzag path to any member of S.

Only-If Part: We will show that if a zigzag path exists between any two checkpoints in S (including a zigzag cycle from a checkpoint to itself), then they cannot belong to the same consistent snapshot. Assume that a zigzag path exists from A to B (note that we can have $A = B$). We use induction on the number n of messages comprising the path to show that A and B cannot belong to the same consistent snapshot.

Basis (n = 1): A zigzag path consisting of one message is also a causal path, so in this case $A \xrightarrow{\text{HB}} B$, implying that they cannot belong to the same consistent shapshot.

Induction: Assume that a zigzag path of n messages $m_1 \cdots m_n$ from one checkpoint to another implies that they cannot belong to the same consistent snapshot. We will show that a zigzag path of $n+1$ messages $m_1 \cdots m_n m_{n+1}$ from A to B implies that they cannot belong to the same consistent snapshot.

To establish a contradiction, assume that some consistent snapshot exists containing both A and B (note that A and B can be the same checkpoint, meaning that a zigzag cycle exists). Let C be the checkpoint immediately following the receive of m_n; then $m_1 \cdots m_n$ constitute a zigzag path from A to C. By the inductive hypothesis, A cannot belong in the same consistent snapshot with C or with any checkpoint following C in the same process. Thus, for A and B to belong to some consistent snapshot, the snapshot must include a checkpoint in C's process that precedes C. However, by the definition of a zigzag path, message m_{n+1} must be sent *after* any such checkpoint, meaning that any checkpoint preceding C happens before B. Thus, B cannot be combined with any checkpoint preceding C to form a consistent snapshot. Therefore, no checkpoint in C's process is available to be combined with both A and B to form a consistent snapshot. QED

C. Examples

The Consistency Theorem shows that the zigzag notion captures exactly the conditions for an arbitrary set of checkpoints to all belong to the same consistent snapshot. Below we consider two interesting special cases. One is the conditions under which an arbitrary checkpoint can be useful in the sense that any of the execution's checkpoints can be combined to form some consistent snapshot to which it belongs. The other is the conditions for two arbitrary checkpoints to both belong to the same consistent snapshot. We illustrate these corollaries with examples.

Corollary 1: A checkpoint C can belong to a consistent snapshot iff C is involved in no zigzag cycle.

In Fig. 2, there is a zigzag path from $C_{2,2}$ to itself (formed by messages $m4$ and $m3$). This zigzag cycle implies that

$C_{2,2}$ cannot belong to *any* consistent shapshot. Because of $m3$ (the message that completes the zigzag cycle), $C_{2,2}$ is inconsistent with $C_{1,1}$ or any preceding checkpoint in process $p1$. Similarly, the Consistency Theorem shows that the zigzag path from $C_{2,2}$ to $C_{1,2}$ (consisting only of message $m4$) means $C_{2,2}$ cannot be in any consistent shapshot with $C_{1,2}$ or any subsequent checkpoint in $p1$. Thus, $C_{2,2}$ cannot be combined with any checkpoint in $p1$ to form a consistent shapshot. Conversely, as illustrated below, if a checkpoint is involved in no zigzag path, then it is always possible to include the checkpoint in a consistent snapshot.

Corollary 2: Checkpoints A and B (belonging to different processes) can belong to the same consistent global snapshot iff

1) no zigzag cycle involving A or B exists, and
2) no zigzag path exists between A and B.

Corollary 2 states that, for two checkpoints to belong together in the same consistent snapshot, no zigzag path can exist between the checkpoints, including a zigzag path from either checkpoint to itself (otherwise, by Corollary 1, they could not belong to any consistent snapshot). In Fig. 2, there is no zigzag path between $C_{1,1}$ and $C_{2,1}$, and neither is involved in a zigzag cycle. Thus, a consistent snapshot exists in which both $C_{1,1}$ and $C_{2,1}$ appear. We can construct such a snapshot, G, by including the first checkpoint in process $p3$ that has no zigzag path to either $C_{1,1}$ or $C_{2,1}$. $C_{3,2}$ is such a checkpoint, since $C_{3,1}$ is the last checkpoint in $p3$ that has a zigzag path to either $C_{1,1}$ (formed by $m2$) or $C_{2,1}$ (formed by $m2$ and $m1$). Note that a zigzag path is formed from $C_{3,1}$ to $C_{2,1}$ by messages $m2$ and $m1$, since $m2$ is received in the same checkpoint interval as $m1$ is sent. Thus, G is $\{C_{1,1}, C_{2,1}, C_{3,2}\}$, shown by the dotted line in Fig. 2. Several observations show that all checkpoints in G are mutually unordered and thus G is consistent. First, no zigzag paths between $C_{1,1}$ and $C_{2,1}$ means that neither happens before the other. Second, no zigzag path from $C_{3,2}$ to either $C_{1,1}$ or $C_{2,1}$ ensures that $C_{3,2}$ does not happen before either of these checkpoints. Finally, neither $C_{1,1}$ or $C_{2,1}$ can happen before $C_{3,2}$; otherwise, a zigzag path would exist from one of them to the other, contradicting the assumption. For example, if $C_{1,1}$ happened before $C_{3,2}$, then the messages comprising this causal path, plus message $m2$, would form a zigzag cycle from $C_{1,1}$ back to itself.

IV. Conclusion

We have shown that the zigzag notion, as a generalization of Lamport's happened-before relation, captures exactly the conditions for a set of checkpoints to belong to the same consistent global snapshot. We proved in this paper that a set of checkpoints can belong to the same consistent global snapshot iff no zigzag path exists from a checkpoint to any other.

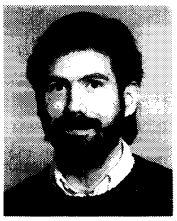
The results can be used in applications that require saving or analyzing consistent snapshots. As mentioned earlier, we have shown in a previous paper one application of our results to reducing rollback propagation for independent checkpointing. We envision the results can also be used in other applications, such as analyzing global states, and reducing space overhead for message logs and checkpoints.

Acknowledgment

The authors thank Y.-M. Wang (Bell Laboratories) for discussions about the results and the anonymous referees for their excellent suggestions for improvement.

References

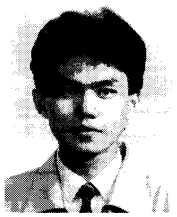
[1] B. Bhargava and S. R. Lian, "Independent checkpointing and concurrent rollback for recovery—An optimistic approach," in *Proc. IEEE Symp. Reliable Distrib. Syst.*, 1988, pp. 3–12.
[2] K. M. Chandy and C. V. Ramamoorthy, "Rollback and recovery strategies for computer programs," *IEEE Trans. Comput.*, vol. 21, pp. 546–556, June 1972.
[3] D. R. Jefferson, "Virtual time," *ACM Trans. Programming Languages, Syst.*, vol. 7, no. 3, pp. 404–425, July 1985.
[4] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. Software Eng.*, vol. SE-13, no. 1, pp. 23–32, Jan. 1987.
[5] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, July 1978.
[6] R. H. B. Netzer and J. Xu, "Adaptive message logging for incremental replay of message-passing programs," in *Proc. Supercomputing '93* Portland, OR, Nov. 1993, pp. 840–849.
[7] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Eng.*, vol. SE-1, no. 2, pp. 220–232, June 1975.
[8] Y. M. Wang, A. Lowry, and W. K. Fuchs, "Consistent global checkpoints based on direct dependency tracking," Res. Rep. RC 18465, IBM T. J. Watson Research Center, Yorktown Heights, NY, Oct. 1992.
[9] Y. M. Wang and W. K. Fuchs, "Optimistic message logging for independent checkpointing in message-passing systems," in *Proc. 11th IEEE Symp. Reliable Distrib. Syst.*, Oct. 1992, pp. 147–154.
[10] L. D. Wittie, "Debugging distributed C programs by real time replay," *SIGPLAN/SIGOPS Workshop Parallel, Distrib. Debugging* Madison, WI, May 1988, pp. 57–67.
[11] J. Xu and R. H. B. Netzer, "Adaptive independent checkpointing for reducing rollback propagation," in *Proc. 5th IEEE Symp. Parallel, Distrib. Processing*, Dec. 1993, pp. 754–761.



Robert Netzer received the B.S.E. degree from the University of Florida, Gainesville, and the M.S. and Ph.D. degrees from the University of Wisconsin-Madison (where he also pursued graduate studies in civil engineering), all in computer science.

He is currently an Assistant Professor in Computer Science at Brown University, Providence, RI. His research focuses on programming tools, especially tools for debugging, fault tolerance, and programming very large networks of heterogenous machines.

Prof. Netzer is a member of the ACM, the IEEE Computer Society, and the ASCE.



Jian Xu received the B.S. and M.S. degrees in 1985 and 1988, respectively, both in computer science, from Shanghai Fudan University. He is working toward the Ph.D. degree in computer science at Brown University, Providence, RI.

His research interests include fault-tolerant computing, parallel and distributed debugging, and programming environments.

He is a student member of the ACM.