Finding Consistent Global Checkpoints in a Distributed Computation

D. Manivannan, *Student Member, IEEE*, Robert H. B. Netzer, *Member, IEEE*, and Mukesh Singhal, *Member, IEEE*

Abstract—Consistent global checkpoints have many uses in distributed computations. A central question in applications that use consistent global checkpoints is to determine whether a consistent global checkpoint that includes a given set of local checkpoints can exist. Netzer and Xu [16] presented the necessary and sufficient conditions under which such a consistent global checkpoint can exist, but they did not explore what checkpoints could be constructed. In this paper, we prove exactly which local checkpoints can be used for constructing such consistent global checkpoints. We illustrate the use of our results with a simple and elegant algorithm to enumerate all such consistent global checkpoints.

Index Terms—Causality, distributed checkpointing, consistent global states, failure recovery, fault tolerance.

1 Introduction

Consistency of global states is a recurring theme in distributed systems. A *global state* is a set of individual process states, one per process, that represents a "snapshot" at some instant of each process's execution. When global states are periodically recorded or analyzed during execution, they are called global *checkpoints*. Global checkpoints have applications in many problems [11], such as transparent failure recovery [10], distributed debugging [7], [8], monitoring distributed events [17], setting distributed breakpoints [15], protocol specification and verification [9], deadlock recovery [12], and others [18], [19]. In this paper, we explore a type of global checkpoint that is said to be *consistent*. A consistent checkpoint is a snapshot of process states that actually occurred simultaneously during the execution or had the potential of doing so [6].

Netzer and Xu [16] proved under what conditions a given set of local checkpoints can be combined with those from other processes to form a global checkpoint that is consistent. These conditions are useful in many algorithms and protocols that record on-the-fly only consistent checkpoints or determine postmortem which global checkpoints are consistent.

To illustrate the problem, consider how to build a consistent checkpoint from an arbitrary set S of local checkpoints drawn from some (but not all) processes. To make the checkpoint global, we must also select one checkpoint from each process not represented in S. Fig. 1 illustrates that a careful choice must be made (a three-process execution is shown; the ×s are local checkpoints). Given two checkpoints such as A and C, if they are to belong to the same consistent checkpoint, consistency [6] requires that neither causally affect the other; this condition is *necessary* for A and C to be consistent. In Fig. 1, a casual path exists from A to C, so any global checkpoint constructed from them will never be consistent. In contrast, checkpoints B and C have no causal path between them (they are mutually *unordered*) and indeed they can be combined with the second checkpoint of P_2 to form a consistent checkpoint, shown by the dashed line. One might be tempted to conclude that *any* set of mutually unordered local checkpoints can always be used to build a consistent checkpoint, but this is not true. Being unordered is not always sufficient to ensure consistency. In Fig. 1, B and D are also unordered, but they *cannot* be combined in a consistent way. There is no checkpoint in P_2 that can be combined with them both while maintaining consistency.

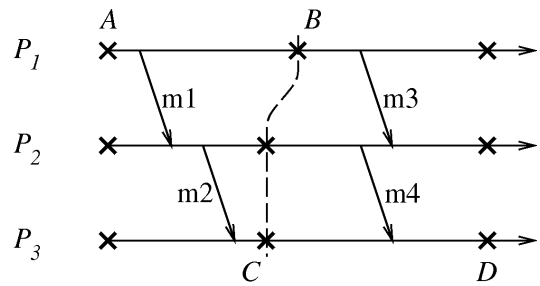


Fig. 1. Example execution: B and C can be used to build a consistent checkpoint (dashed line) but A and C, or B and D, cannot.

In this paper, we prove precisely which local checkpoints from each process can be combined with those from a given set S to build a global checkpoint that is consistent. Although Netzer and Xu [16] proved the conditions necessary and sufficient for *some* such consistent checkpoint to exist, they did not explore how to construct it. Building on Netzer and Xu's results, Wang [18], [19] presents algorithms

- *D. Manivannan and M. Singhal are with the Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210. E-mail: {manivann, singhal}@cis.ohio-state.edu.*
- *R.H.B. Netzer is with the Department of Computer Science, Brown University, Box 1910, Providence, RI 02912. E-mail: rn@cs.brown.edu.*

Manuscript received 29 Mar. 1996.
For information on obtaining reprints of this article, please send e-mail to: transtpds@computer.org, and reference IEEECS Log Number 100160.

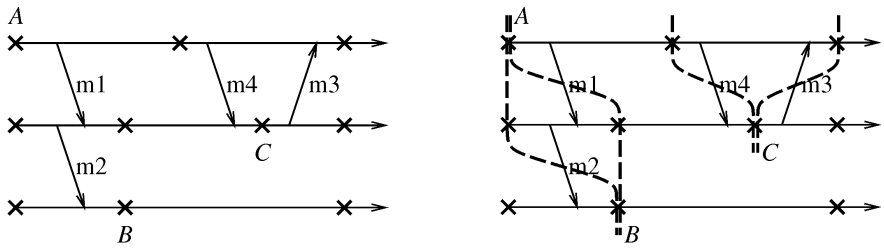


Fig. 2. Z-paths: (a) Z-path from A to B and Z-cycle involving C; (b) any cut through A and B is inconsistent, as well as any cut involving C (all the dashed lines are inconsistent).

for computing the minimal and maximal consistent checkpoints that contain S. We further build on this past work by proving exactly which sets of local checkpoints can be combined with those in S and still retain consistency. We illustrate the results with an algorithm that enumerates all such consistent checkpoints, and show how the minimum and maximum checkpoints are special cases.

2 Model

We model a distributed computation as having N processes, $P_1, P_2, \cdots, P_N$, between which messages are delivered reliably but with arbitrary delay. Execution of a process is modeled by three types of events: the send of a message, the receive of a message, and a (local) checkpoint. We use Lamport's *happened before* relation [13], $\xrightarrow{\text{HB}}$, defined over this set of events. If $a \xrightarrow{\text{HB}} b$, we say that a *causal path* exists from a to b. If neither happened before the other, we call them *unordered*.

A *local checkpoint* of a process is a distinguished state that happens to be of interest in the problem under consideration. States of interest could be points at which a local predicate becomes true, where a deadlock is detected, or where the process state is checkpointed to disk. Local checkpoints are given as part of the problem; we have no freedom to arbitrarily place them (just as we have no control over where a predicate becomes true). A set of local checkpoints, one from each process, is called a *consistent global checkpoint* (or just a *consistent checkpoint*) iff none happened before any other in the set. The ith ($i \geq 0$) checkpoint of process P_p is denoted $C_{p,i}$. We assume that each process takes a checkpoint before execution begins and after execution ends. We sometimes denote checkpoints by the letters A, B, or C for clarity. The ith *checkpoint interval* of process P_p is all the computation performed between its $(i-1)$th and ith checkpoints (and includes $C_{p,i-1}$ but not $C_{p,i}$).

3 Background and Related Work

The definition of consistency states that for a set S of local checkpoints to be a consistent checkpoint, S must contain one local checkpoint from *each* of the N processes, and no causal path should exist between any two checkpoints in S. However, when $|S| < N$, having no causal paths between the checkpoints in S is insufficient by itself to ensure that local checkpoints from processes not represented in S can be combined with S to form a global checkpoint that is consistent. Netzer and Xu [16] define a generalization of causal paths called *zigzag paths* (which we call *Z-paths* for brevity) and prove that the absence of Z-paths between checkpoints in S is exactly the condition that guarantees S can be extended to a consistent checkpoint. Because Z-paths express the exact conditions for consistency, they are a powerful notion for reasoning about consistent states and have been applied recently in several problems [2], [3], [4], [5].

Definition 1. *A Z-path exists from $C_{p,i}$ to $C_{q,j}$ iff*

1) *$p = q$ and $i < j$ (i.e., one checkpoint precedes the other in the same process), or*[1]
2) *there exist messages $m_1, m_2, \cdots, m_n$ ($n \geq 1$) such that*
   a) *m_1 is sent by process P_p after $C_{p,i}$,*
   b) *if m_k ($1 \leq k < n$) is received by P_r, then m_{k+1} is sent by P_r in the same or a later checkpoint interval (m_{k+1} may be sent before or after m_k is received), and*
   c) *m_n is received by P_q before $C_{q,j}$.*

A checkpoint C is said to be in a Z-cycle iff there exists a Z-path from C to itself.

A Z-path between two checkpoints A and B is like a causal path—both are sequences of messages that start after A and end before B, and both define a transitive relation—but their differences are important.

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. Such a chain is also a Z-path, but a Z-path is also allowed to have any message in the chain 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 Z-path, but a Z-path may not be a causal path. Fig. 2a illustrates this difference. There is a Z-path from A to B because message $m1$ is sent after A, $m2$ is sent in the *same* checkpoint interval as $m1$ is received, and $m2$ is received before B (the path forms a zigzag shape, hence the name). This Z-path is not a causal path.

Another difference stems from Z-paths not always defining a partial order. A Z-path can exist from a checkpoint back to itself (a *Z-cycle*). In contrast, causal paths never form cycles. In Fig. 2a, a Z-cycle exists involving checkpoint C. Message $m3$ is sent after C, $m4$ is sent in the same interval in which $m3$ is received, and $m4$ is received before C, completing the Z-path from C to itself (the Z-cycle).

1. Netzer and Xu's definition only contains the second clause, but it is convenient to also define a Z-path to exist from A to B if A and B belong to the same process and A precedes B.

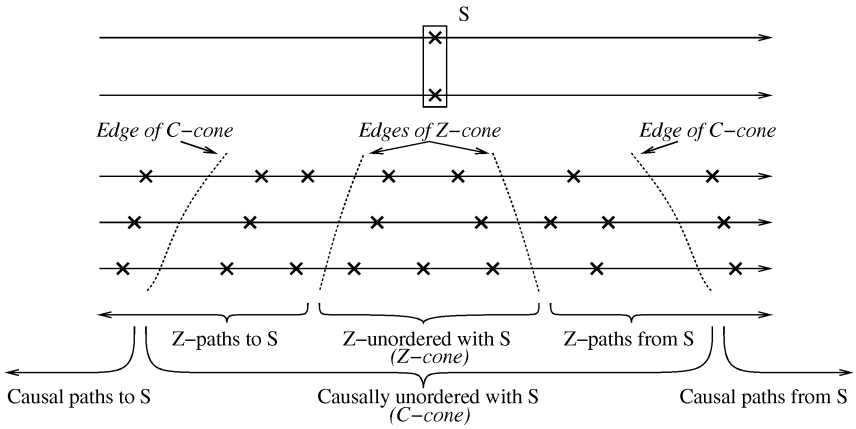


Fig. 3. The *Z-cone* and the *C-cone* associated with a set of checkpoints *S*.

To reason about Z-paths, we use the following notation, motivated by Wang [18], [19].

DEFINITION 2. *Let A, B be individual checkpoints and R, S be sets of checkpoints. Let* $\rightsquigarrow$ *be a relation defined over checkpoints and sets of checkpoints such that*

1) $A \rightsquigarrow B$ *iff a Z-path exists from A to B,*
2) $A \rightsquigarrow S$ *iff a Z-path exists from A to* some *member of S,*
3) $S \rightsquigarrow A$ *iff a Z-path exists from* some *member of S to A, and*
4) $R \rightsquigarrow S$ *iff a Z-path exists from* some *member of R to* some *member of S.*

Using this notation, the results of Netzer and Xu are easily stated. Note that $S \not\rightsquigarrow S$ implies the checkpoints in S are all from different processes. ($\not\rightsquigarrow$ denotes not $\rightsquigarrow$.)

THEOREM 1. *A set of checkpoints S can be extended to a consistent checkpoint if and only if* $S \not\rightsquigarrow S$.

COROLLARY 1. *A checkpoint C can be part of a consistent checkpoint if and only if it is not involved in a Z-cycle.*

4 FINDING CONSISTENT CHECKPOINTS

Although Netzer and Xu prove the exact conditions under which a set of checkpoints S can be used to build a consistent checkpoint, they do not discuss how to actually construct the consistent checkpoints containing S. Our results concern this issue. Given a set S of checkpoints such that $S \not\rightsquigarrow S$, we analyze what other local checkpoints can be combined with S to build a consistent checkpoint.

There are three important observations. First, none of the checkpoints that have a Z-path to or from any of the checkpoints in S can be used because, by Theorem 1, no checkpoints between which a Z-path exists can ever be part of a consistent checkpoint. Thus, only those checkpoints that have no Z-paths to **or** from any member of S are candidates. We call the set of all such candidates the *Z-cone* of S. Similarly, we call the set of all checkpoints that have no causal path to or from any checkpoint in S the *C-cone* of S.[2]

2. These terms are inspired by the so-called *light cone* of an event e which is the set of all events with causal paths from e (i.e., events in e's future) [14]. Although the light cone of e contains events ordered *after* e, we define the Z-cone and C-cone of S to be those events with *no* zigzag or causal ordering, respectively, to or from any member of S.

DEFINITION 3. *Let S be a set of checkpoints. The Z-Cone of S is defined as*

$$Z\text{-}Cone(S) = \{A \mid S \not\rightsquigarrow A \wedge A \not\rightsquigarrow S\}.$$

The Z-cone helps us reason about orderings and consistency. To build a consistent checkpoint from S, we must draw local checkpoints only from $Z\text{-}Cone(S)$.

LEMMA 1. *Let S be a set of checkpoints. If* $S \not\rightsquigarrow S$, *then Z-Cone(S) contains at least one checkpoint from each process (and* $S \subseteq$ *Z-Cone(S)).*

PROOF. From Theorem 1, S can be extended to a consistent checkpoint, so every process has at least one checkpoint that can be used for constructing a consistent checkpoint containing S. All such checkpoints have no Z-path from or to any of the checkpoints in S. By Definition 3, all such checkpoints belong to Z-Cone(S). In particular, $S \subseteq Z\text{-}Cone(S)$. □

Since causal paths are always Z-paths, the Z-cone of S is a subset of the C-cone of S, illustrated in Fig. 3 for an arbitrary S. Note that if a Z-path exists from checkpoint $C_{q,j}$ in process P_q to a checkpoint in S, then a Z-path also exists from every checkpoint in P_q preceding $C_{q,j}$ to the same checkpoint in S because Z-paths are transitive (causal paths are transitive as well).

Our second observation is that, although candidates for building a consistent checkpoint from S must lie in the Z-cone of S, not all checkpoints in the Z-cone are usable. If a checkpoint in $Z\text{-}cone(S)$ is involved in a Z-cycle, then, by Corollary 1, it cannot be part of a consistent checkpoint that includes S. We prove below that if we remove from consideration all such checkpoints, then the remaining ones are exactly those that are "useful" in the sense that each can be individually used to build *some* consistent checkpoint that includes S.

LEMMA 2. *Let S be a set of checkpoints such that* $S \not\rightsquigarrow S$. *Let A be any checkpoint not already in S. Then,* $S \cup \{A\}$ *can be extended to a consistent checkpoint if and only if* $(A \in Z\text{-}Cone(S)) \wedge (A \not\rightsquigarrow A)$.

PROOF. $S \cup \{A\}$ can be extended to a consistent checkpoint $\Leftrightarrow (S \cup \{A\}) \not\rightsquigarrow (S \cup \{A\})$ by Theorem 1 $\Leftrightarrow ((S \not\rightsquigarrow A) \wedge (A \not\rightsquigarrow S) \wedge (A \not\rightsquigarrow A)) \Leftrightarrow ((A \in \text{Z-Cone}(S)) \wedge (\text{A} \not\rightsquigarrow \text{A}))$. □

Lemma 2 states that, if we are given a set S such that $S \rightsquigarrow S$, we are guaranteed that any *single* checkpoint from the Z-Cone that is not on a Z-cycle can belong to *some* consistent checkpoint that also contains S. However, our final observation is that, if we attempt to build a consistent checkpoint from S by choosing any *subset* T of checkpoints from Z-Cone(S) to combine with S, we have no guarantee that the checkpoints in T have no Z-paths among them. In other words, Z-paths may still exist between members of Z-Cone(S). Intuitively, we cannot "thread" just any line through the Z-cone and expect it to be consistent with S, even if it avoids Z-cycles. We have one final constraint we must place on the set T: The checkpoints in T must have no Z-paths among them. Furthermore, since $S \rightsquigarrow S$ by Theorem 1, at least one such T must exist.

THEOREM 2. *Let S be a set of checkpoints such that $S \rightsquigarrow S$ and let T be any set of checkpoints such that $S \cap T = \emptyset$. Then, $S \cup T$ is a consistent checkpoint if and only if*

1) $T \subseteq$ *Z-Cone*(S),
2) $T \rightsquigarrow T$, *and*
3) $|S \cup T| = N$.

PROOF. $S \cup T$ is a consistent checkpoint $\Leftrightarrow ((S \cup T) \rightsquigarrow (S \cup T)) \wedge (|S \cup T| = N)$ by Theorem 1 and the definition of consistency $\Leftrightarrow ((T \rightsquigarrow S) \wedge (S \rightsquigarrow T) \wedge (T \rightsquigarrow T) \wedge (S \rightsquigarrow S) \wedge (|S \cup T| = N)) \Leftrightarrow ((T \subseteq \text{Z-Cone}(S)) \wedge (T \rightsquigarrow T) \wedge (S \rightsquigarrow S) \wedge (|S \cup T| = N))$. □

To briefly illustrate an application of our results, we next consider two cases of finding consistent checkpoints. The first is an algorithm to enumerate the set of all consistent checkpoints that include S. We are not suggesting any particular application of this algorithm, but consistent-state exploration has been used in past work [1], [6] and it is illustrative of other techniques which must understand the structure of the set of consistent checkpoints. Our results show the structure of this set, and our algorithm is novel in that it restricts its search only to those combinations of checkpoints within the Z-cone of S which can be combined together in a consistent way. We avoid exhaustive searching for finding consistent checkpoints by selecting only local checkpoints that we know will be useful, as defined next.

DEFINITION 4. *Let S be a set of checkpoints such that $S \rightsquigarrow S$. For each process P_q, define $USEFUL_q(S) = \{C_{q,i} \mid C_{q,i} \in$ Z-Cone(S) $\wedge\ C_{q,i} \rightsquigarrow C_{q,i}\}$.*

Our algorithm is shown in Fig. 4. The function *ComputeAllCgs*(S) returns the set of all consistent checkpoints that contain S. The crux of our algorithm is the function *ComputeAllCgsFrom*(T, *ProcSet*) which extends a set of checkpoints T in all possible consistent ways, but uses checkpoints only from processes in the set *ProcSet*. After verifying that $S \rightsquigarrow S$, *ComputeAllCgs* simply calls *ComputeAllCgsFrom*, passing a *ProcSet* consisting of the processes not represented in S (lines 2–5). The resulting consistent checkpoints are collected in the global variable G which is returned (line 6). It is worth noting that if $S = \emptyset$, the algorithm computes *all* of the consistent checkpoints that exist in the execution.

```
1. ComputeAllCgs(S) {
2.   let G = ∅
3.   if S ⇝ S then
4.     let AllProcs be the set of all
       processes not represented in S;
5.     ComputeAllCgsFrom(S, AllProcs);
6.   return G
7. }
8. ComputeAllCgsFrom(T, ProcSet) {
9.   if (ProcSet = ∅) then
10.    G = G ∪ {T}
11.  else
12.    let P_q be any process in ProcSet;
13.    for each checkpoint C ∈ USEFUL_q(T)
       do
14.      ComputeAllCgsFrom(T∪{C},ProcSet\{P_q})
15.}
```

Fig. 4. Algorithm for computing all consistent checkpoints containing S.

The recursive function *ComputeAllCgsFrom*(T, *ProcSet*) chooses a process from *ProcSet*, say P_q, and iterates through all checkpoints C in $USEFUL_q(T)$. Recall that Lemma 2 states that each such checkpoint extends T part-way toward a consistent checkpoint. This means that $T \cup \{C\}$ can itself be further extended, eventually arriving at a consistent checkpoint. Since this further extension is simply another instance of constructing all consistent checkpoints that contain checkpoints from a given set, we make a recursive call (line 14), passing $T \cup \{C\}$ and a *ProcSet* from which P_q is removed. The recursion terminates when the passed set contains checkpoints from all processes (i.e., *ProcSet* is empty). In this case, T is finally a global checkpoint and it is added to G (line 10). When the algorithm terminates, all candidates in Z-Cone(S) that are not on Z-cycles have been used in extending S, so G contains every possible consistent checkpoint that contains S.

Many variants of this algorithm are possible, such as those that compute certain consistent checkpoints without enumerating them all. Such algorithms must effectively compute the Z-cone, which reduces to determining which Z-paths exist in the execution. Tracking Z-paths on-the-fly is difficult and currently remains an open problem, although subsets of Z-paths sufficient to detect parts of the Z-cone can be easily detected [3], [20]. Finding Z-paths postmortem is straightforward by computing a linear-time transformation of the directed graph representing the execution; Wang [18], [19] defines a graph called the *rollback-dependency* graph (or *R-graph*) which shows Z-paths in a distributed computation that has terminated or stopped execution. It is easy to find Z-paths and Z-cones from such a graph.

An interesting variant of the algorithm computes the consistent *minimal* and *maximal* checkpoints that contain S. Intuitively, these are the "earliest" and "latest" consistent checkpoints containing S that can be constructed. The following is based on Wang [18], [19].

DEFINITION 5. *Let S be any set of checkpoints such that $S \rightsquigarrow S$. Let $M = S \cup T$ be a consistent checkpoint, where $T = \left\{C_{p_1,i_1}, C_{p_2,i_2}, \cdots C_{p_k,i_k}\right\}$, and $T \cap S = \emptyset$. Then,*

1) *M is the maximal consistent checkpoint containing S iff for any consistent checkpoint M′ = S ∪ T′, where T′ =* $\{C_{p_1,j_1}, C_{p_2,j_2}, \cdots C_{p_k,j_k}\}$, *we have* $\forall n : 1 \le n \le k.\ i_n \ge j_n$.
2) *M is the* minimal *consistent checkpoint containing S iff for any consistent checkpoint M′ = S ∪ T′, where T′ =* $\{C_{p_1,j_1}, C_{p_2,j_2}, \cdots C_{p_k,j_k}\}$, *we have* $\forall n : 1 \le n \le k.\ i_n \le j_n$.

Wang [18], [19] showed that the minimal (maximal) consistent checkpoints containing *S* are those formed by choosing, from each process not represented in *S*, the earliest (latest) checkpoint that has no Z-path to or from any member of *S*. Netzer and Xu [16] construct the minimal checkpoint in one of their proofs but never discuss its properties per se.

In terms of the Z-cone of *S*, the minimal and maximal checkpoints are exactly those drawn through the "leading" and "trailing" edges of the Z-cone. When $S \rightsquigarrow S$, this implies that the Z-cone possesses some interesting properties. The leading and trailing edges *always* exist and never have Z-paths among them (including any Z-cycles) meaning that they are *always* consistent.

5 Conclusion

The two problems above illustrate that our theoretical foundation is helpful for reasoning about consistency. Our characterization of which local checkpoints are useful can find application in algorithms that must piece together consistent global checkpoints from individual local checkpoints. The notion of the Z-cone, and of which checkpoints within the Z-cone are useful, provides a new understanding of the structure of consistent global checkpoints.

References

[1] O. Babaoglu and K. Marzullo, "Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms," *Distributed Systems,* S. J. Mullender, ed., pp. 55–96. Addison-Wesley, 1993.

[2] R. Baldoni, J.M. Helary, A. Mostefaoui, and M. Raynal, "Characterizing Consistent Checkpoints in Large-Scale Distributed Systems," *Proc. Fifth IEEE Int'l Conf. Parallel and Distributed Computing,* pp. 314–323, Chejiu Islands, South Korea, Aug. 1995.

[3] R. Baldoni, J.M. Helary, A. Mostefaoui, and M. Raynal, "Consistent Checkpoints in Message Passing Distributed Systems," Rapporte de Recherche No. 2564, INRIA, France, June 1995.

[4] R. Baldoni, J.M. Helary, A. Mostefaoui, and M. Raynal, "On Modeling Consistent Checkpoints and the Domino Effect in Distributed Systems," Rapporte de Recherche No. 2569, INRIA, France, June 1995.

[5] R. Baldoni, J.M. Helary and M. Raynal, "About Recording in Asynchronous Computations," *Proc. 15th ACM Symp. Principles of Distributed Computing,* p. 55, May 1996.

[6] K.M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Trans. Computer Systems,* vol. 3, no. 1, pp. 63–75, Feb. 1985.

[7] R. Cooper and K. Marzullo, "Consistent Detection of Global Predicates," *Proc. ACM/ONR Workshop on Parallel and Distributed Debugging,* pp. 163–173, May 1991.

[8] E. Fromentin, N. Plouzeau, and M. Raynal, "An Introduction to the Analysis and Debug of Distributed Computations," *Proc. First IEEE Int'l Conf. Algorithms and Architectures for Parallel Processing,* pp. 545–554, Brisbane, Australia, Apr. 1995.

[9] K. Geihs and M. Seifert, "Automated Validation of a Co-operation Protocol for Distributed Systems," *Proc. Sixth Int'l Conf. Distributed Computing Systems,* pp. 436–443, 1986.

[10] R. Koo and S. Toueg, "Checkpointing and Roll-back Recovery for Distributed Systems," *IEEE Trans. Software Eng.,* vol. 13, no. 1, pp. 23–31, Jan. 1987.

[11] A.D. Kshemkalyani, M. Raynal, and M. Singhal, "An Introduction to Snapshot Algorithms in Distributed Computing," *Distributed Systems Eng. J.,* vol. 2, no. 4, pp. 224–233, Dec. 1995.

[12] A.D. Kshemkalyani and M. Singhal, "Efficient Detection and Resolution of Generalized Distributed Deadlocks," *IEEE Trans. Software Eng.,* vol. 20, no. 1, pp. 43–54, Jan. 1994.

[13] L. Lamport, "Time, Clocks and Ordering of Events in Distributed Systems," *Comm. ACM,* vol. 21, no. 7, pp. 558–565, July 1978.

[14] L. Lamport, "The Mutual Exclusion Problem: Part I—A Theory of Interprocess Communication," *J. ACM,* vol. 33, no. 2, pp. 313–326, Apr. 1986.

[15] B. Miller and J. Choi, "Breakpoints and Halting in Distributed Programs," *Proc. Eighth Int'l Conf. Distributed Computing Systems,* pp. 316–323, 1988.

[16] R.H.B. Netzer and J. Xu, "Necessary and Sufficient Conditions for Consistent Global Snapshots," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, no. 2, pp. 165–169, Feb. 1995.

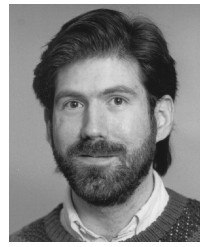
[17] M. Spezialetti and P. Kearns, "Simultaneous Regions: A Framework for the Consistent Monitoring of Distributed Systems," *Proc. Ninth Int'l Conf. Distributed Computing Systems,* pp. 61–68, 1989.

[18] Y.-M. Wang, "Maximum and Minimum Consistent Global Checkpoints and their Applications," *Proc. 14th IEEE Symp. Reliable Distributed Systems,* pp. 86–95, Bad Neuenahr, Germany, Sept. 1995.

[19] Y.-M. Wang, "Consistent Global Checkpoints that Contain a Given Set of Local Checkpoints," *IEEE Trans. Computers,* vol. 46, no. 4, pp. 456–468, Apr. 1997.

[20] J. Xu and R.H.B. Netzer, "Adaptive Independent Checkpointing for Reducing Rollback Propagation," *Proc. Fifth IEEE Symp. Parallel and Distributed Processing,* pp. 754–761, Dec. 1993.

D. Manivannan received his BSc degree in mathematics from the University of Madras, India. He received an MS degree in mathematics and an MS degree in computer science from the Ohio State University, Columbus, Ohio, in 1992 and 1993, respectively. He is currently a PhD student in the Department of Computer and Information Science, the Ohio State University, Columbus. His research interests include distributed systems, mobile computing systems, and interprocessor communication in parallel architectures. He is a student member of the ACM and the IEEE Computer Society.



Robert Netzer received the BSE degree in computer science from the University of Florida, and the MS and PhD degrees, both in computer science, from the University of Wisconsin-Madison, where he also pursued graduate studies in civil engineering.

Dr. Netzer is currently an assistant professor in computer science at Brown University. His research addresses the instrumentation of systems for a wide variety of problems, with a focus on programming and debugging tools. Professor Netzer is a member of the ACM and the IEEE Computer Society.

Mukesh Singhal received a BE degree in electronics and communication engineering with high distinction from the University of Roorkee, Roorkee, India, in 1980, and a PhD degree in computer science from the University of Maryland, College Park, in May 1986. He is an associate professor of computer and information science at the Ohio State University, Columbus. His current research interests include operating systems, distributed systems, mobile computing, high-speed networks, and performance modeling. He has published more than 85 refereed articles in these areas.

Dr. Singhal has coauthored two books, *Advanced Concepts in Operating Systems* (McGraw-Hill, New York, 1994) and *Readings in Distributed Computing Systems* (IEEE CS Press, 1993). He is an editor of the IEEE CS Press.