# Quasi-Synchronous Checkpointing: Models, Characterization, and Classification

D. Manivannan, *Member*, *IEEE*, and Mukesh Singhal, *Senior Member*, *IEEE*

**Abstract**—Checkpointing algorithms are classified as *synchronous* and *asynchronous* in the literature. In *synchronous* checkpointing, processes synchronize their checkpointing activities so that a globally consistent set of checkpoints is always maintained in the system. Synchronizing checkpointing activity involves message overhead and process execution may have to be suspended during the checkpointing coordination, resulting in performance degradation. In *asynchronous* checkpointing, processes take checkpoints without any coordination with others. Asynchronous checkpointing provides maximum autonomy for processes to take checkpoints; however, some of the checkpoints taken may not lie on any consistent global checkpoint, thus making the checkpointing efforts useless. Asynchronous checkpointing algorithms in the literature can reduce the number of useless checkpoints by making processes take communication induced checkpoints besides asynchronous checkpoints. We call such algorithms *quasi-synchronous*. In this paper, we present a theoretical framework for characterizing and classifying such algorithms. The theory not only helps to classify and characterize the quasi-synchronous checkpointing algorithms, but also helps to analyze the properties and limitations of the algorithms belonging to each class. It also provides guidelines for designing and evaluating such algorithms.

**Index Terms**—Causality, distributed checkpointing, consistent global checkpoint, failure recovery, fault tolerance, zigzag paths.

## 1 Introduction

During the execution of a distributed computation, processes exchange information via messages. The message exchange establishes causal dependencies among states of processes.[1] The causal dependency among the states of processes is formally characterized by Lamport's "happened before" relation [11]. Informally, a state $s_q$ of a process $P_q$ is causally dependent on a state $s_p$ of another process $P_p$ if a message (or sequences of messages) sent by $P_p$ after state $s_p$ was received by $P_q$ before reaching state $s_q$.

A local checkpoint of a process is a recorded state of the process. A set of local checkpoints, one from each of the processes involved in a distributed computation, is called a *consistent global checkpoint* if none of them is causally dependent on any other checkpoint in the set. Determining consistent global checkpoints has applications in several areas of distributed system design [9]. Some areas of application are failure recovery, debugging distributed software, monitoring distributed events such as in industrial process control, setting distributed breakpoints, and protocol specification and verification.

In the literature, several checkpointing schemes have been proposed for distributed systems. These schemes are generally classified into two categories—*synchronous* and *asynchronous*. In *synchronous* checkpointing schemes, processes synchronize their checkpointing activities so that a globally consistent set of checkpoints is always maintained in the system [6], [8], [12]. The storage requirement for the checkpoints is minimum because each process needs to keep at most two checkpoints (one committed and one possibly not committed) in stable storage at any given time. Major disadvantages of synchronous checkpointing are 1) process execution may have to be suspended during the checkpointing coordination, as in [8], resulting in performance degradation and 2) it requires extra message overhead to synchronize the checkpointing activity.

In *asynchronous* checkpointing [4], [10], processes take local checkpoints periodically without any coordination with each other. This approach allows maximum process autonomy for taking checkpoints and has no message overhead for local checkpointing. A process determines consistent global checkpoints by communicating with other processes to determine the dependency among local checkpoints.

In asynchronous checkpointing, it could very well happen that processes took checkpoints such that none of the checkpoints lies on a consistent global checkpoint. A local checkpoint that cannot be part of a consistent global checkpoint is said to be *useless*. A local checkpoint that can be part of a consistent global checkpoint is called an *useful* checkpoint. Fig. 1 illustrates a distributed computation in which two processes take checkpoints asynchronously. Note that none of the checkpoints taken is useful and all checkpointing effort is wasted. Fig. 1 shows the worst case scenario; however, in general, a number of checkpoints will be useful.

The number of useless checkpoints taken by processes can be reduced by requiring processes to take communication induced checkpoints in addition to checkpoints taken independently [15], [24]. The notion of communication

1. The state of a process is characterized by the state of its local memory and a history of activity.

- *D. Manivannan is with the Computer Science Department, University of Kentucky, Lexington, KY 40506. Email: manivann@cs.uky.edu.*
- *M. Singhal is with the Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210. Email: singhal@cis.ohio-state.edu.*

*Manuscript received 4 June 1996; revised 4 May 1997.*
*For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 100220.*

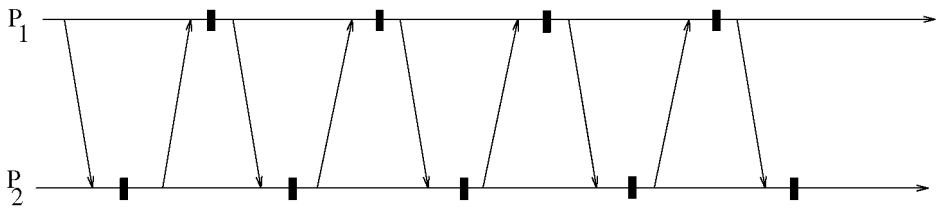


Fig. 1. A distributed computation with asynchronous checkpointing.

induced checkpoints can be traced back to the notion of branch recovery points introduced by Kim [7]. The checkpointing algorithms that require processes to take communication induced checkpoints are called *quasi-synchronous* checkpointing algorithms because some checkpointing activity is triggered by the message pattern and knowledge gained about the dependency between checkpoints of processes. Checkpoints taken by processes independently are called *basic* checkpoints and the communication induced checkpoints are called *forced* checkpoints. Some of the advantages of a quasi-synchronous checkpointing algorithm are that it can reduce the number of useless checkpoints and advance the recovery line. For example, in Fig. 1, if each process took a forced checkpoint prior to receiving every message, then all the checkpoints taken would be useful.

### 1.1 Paper Objectives

Quasi-synchronous checkpointing algorithms are attractive because they can reduce the number of useless checkpoints which, in turn, helps in bounding rollback distance during recovery. We provide a theoretical framework for the characterization and classification of quasi-synchronous checkpointing algorithms. The characterization and the classification provide a deeper understanding of the principles underlying quasi-synchronous checkpointing algorithms, helping us evaluate such algorithms and providing guidelines for designing more efficient checkpointing algorithms. The classification also provides a clear understanding of the properties of the checkpointing algorithms belonging to each class.

The rest of the paper is organized as follows. Section 2 provides the background required for the paper. In Section 3, we provide a characterization of quasi-synchronous checkpointing algorithms. In Section 4, we present a classification of quasi-synchronous checkpointing algorithms. The merits of the classification are discussed in Section 5. Section 6 concludes the paper.

## 2 Preliminaries

### 2.1 System Model

The distributed computation we consider consists of $N$ spatially separated sequential processes denoted by $P_1, P_2, \cdots, P_N$. The processes do not share a common memory or a common clock. Message passing is the only way for processes to communicate with one another. The computation is asynchronous: Each process progresses at its own speed and messages are exchanged through reliable communication channels, whose transmission delays are finite but arbitrary.

Execution of a process is modeled by three types of events—the send event of a message, the receive event of a message, and an internal event. The states of processes depend on one another due to interprocess communication. Lamport's *happened before* relation [11] on events, $\xrightarrow{\text{hb}}$, is defined as the transitive closure of the union of two other relations: $\xrightarrow{\text{hb}} = (\xrightarrow{\text{xo}} \cup \xrightarrow{\text{m}})^+$. The $\xrightarrow{\text{xo}}$ relation captures the order in which local events of a process are executed. The $i$th event of any process $P_p$ (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 relation between the send and receive events of the same message: If $a$ is the send event of a message and $b$ is the corresponding receive event of the same message, then $a \xrightarrow{\text{m}} b$.

Each checkpoint taken by a process is assigned a unique sequence number. The $i$th $(i \geq 0)$ checkpoint of process $P_p$ is assigned the sequence number $i$ and is denoted by $C_{p,i}$. We assume that each process takes an initial checkpoint before execution begins and a *virtual* checkpoint after execution ends. Sometimes, checkpoints are also denoted by the letters $A$, $B$, or $C$ for clarity. The $i$th checkpoint interval of process $P_p$ is all the computation performed between its $(i-1)$th and $i$th checkpoints (and includes the $(i-1)$th checkpoint, but not the $i$th).

The send and the receive events of a message $M$ are denoted respectively by $send(M)$ and $receive(M)$. So, $send(M) \xrightarrow{\text{hb}} C_{p,i}$ if message $M$ was sent by process $P_p$ before taking the checkpoint $C_{p,i}$. Also, $receive(M) \xrightarrow{\text{hb}} C_{p,i}$ if message $M$ was received and processed by $P_p$ before taking the checkpoint $C_{p,i}$. $send(M) \xrightarrow{\text{hb}} receive(M)$ for any message $M$. Next, we present the definition of a consistent global checkpoint formally.

**Definition 1.** *A set* $S = \{C_{1,m_1}, C_{2,m_2}, \cdots, C_{N,m_N}\}$ *of* $N$ *checkpoints, one from each process, is said to be a* consistent global checkpoint[2] *if, for any message* $M$ *and for any integer* $p,\ 1 \leq p \leq N$: $receive(M) \xrightarrow{\text{hb}} C_{p,m_p} \Longrightarrow send(M) \xrightarrow{\text{hb}} C_{q,m_q}$ *for some* $q,\ 1 \leq q \leq N$.

### 2.2 Z-Paths and Their Properties

Netzer and Xu [17] gave a necessary and sufficient condition for a given set of checkpoints to be part of a consistent global checkpoint. They introduced the notion of

2. Also called a **consistent global snapshot** or **a consistent cut.**

*zigzag path*, which is a generalization of a *causal path*[3] induced by Lamport's happened before relation. A zigzag path (or a *Z-path* for short) between two checkpoints is like a causal path, but a Z-path allows a message to be sent before the previous one in the path is received. Formally, a *Z-path* between two checkpoints is defined [14], [17] as:

**Definition 2.** *A Z-path exists from $C_{p,i}$ to $C_{q,j}$ if*

1. *$p = q$ and $i < j$ (i.e., the two checkpoints are from the same process and the former precedes the later) or*
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 later checkpoint interval (although $m_{k+1}$ may be sent before or after $m_k$ is received), and*
   c. *$m_n$ is received by $P_q$ before $C_{q,j}$.*

The following notations are used throughout the paper:

**Definition 3.** *Let A,B be individual checkpoints and R,S be sets of checkpoints. We define the relations $\overset{zp}{\rightsquigarrow}$ and $\overset{cp}{\rightsquigarrow}$ over checkpoints and sets of checkpoints as follows:*

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

*Similarly, $A \overset{cp}{\rightsquigarrow} B$ iff there exists a causal path from A to B, and $A \overset{cp}{\rightsquigarrow} S$ iff a causal path exists from A to some member of S, etc.*

An important property of Z-paths is that it captures the precise requirement for a set of checkpoints to be a part of a consistent global checkpoint, as stated in the following theorem due to Netzer and Xu [17].

**Theorem 1.** *A set of checkpoints $S$ can be extended to a consistent global checkpoint if and only if $S \overset{zp}{\not\rightsquigarrow} S$.*

**Proof.** The proof can be found in [17]. □

Theorem 1 presents the precise condition a set of checkpoints needs to satisfy in order to be part of a consistent global checkpoint. The following corollary of Theorem 1 gives the precise condition for a single checkpoint to be part of a consistent global checkpoint.

**Corollary 1.** *A checkpoint $C$ can be part of a consistent global checkpoint if and only if it is not on a Z-cycle.*

**Proof.** Follows from Theorem 1 by taking $S = \{C\}$. □

We first review some of the results in [13], [14] to make the paper self-contained. Given a set $S$ of checkpoints such that $S \overset{zp}{\not\rightsquigarrow} S$, those checkpoints that have no Z-paths from or to any of the checkpoints in $S$ are possible candidates for extending $S$ to a consistent global checkpoint. The set of all such checkpoints is called the *Z-cone* of $S$. The set of all those checkpoints that have no causal path from or to any of the checkpoints in $S$ is called the *C-cone* of $S$. Clearly, the Z-cone of $S$ is a subset of the C-cone of $S$. The Z-cone and the C-cone of a set of checkpoints $S$ are depicted pictorially in Fig. 2. Formally, the Z-cone and the C-cone of a given set $S$ can be defined as follows:

**Definition 4.** *Let $S$ be a set of checkpoints such that $S \overset{zp}{\not\rightsquigarrow} S$. Then, the Z-cone of S, denoted Z-cone(S), is defined as*

$$Z\text{-}cone(S) = \{C_{q,i} \mid (S \overset{zp}{\not\rightsquigarrow} C_{q,i}) \wedge (C_{q,i} \overset{zp}{\not\rightsquigarrow} S)\}.$$

*Similarly, C-cone(S) is defined as*

$$C\text{-}cone(S) = \{C_{q,i} \mid (S \overset{cp}{\not\rightsquigarrow} C_{q,i}) \wedge (C_{q,i} \overset{cp}{\not\rightsquigarrow} S)\}.$$

Given a set of checkpoints $S$ such that $S \overset{zp}{\not\rightsquigarrow} S$, for each process not represented in $S$, if we include in $S$ the first checkpoint in the Z-cone of $S$, then the resulting set is guaranteed to be the minimal consistent global checkpoint containing $S$. Likewise, for each process not represented in $S$, if we include in $S$ the last checkpoint in the Z-cone of $S$, then the resulting set is guaranteed to be the maximal consistent global checkpoint containing $S$. The dotted lines in Fig. 3 pass through the maximal and the minimal consistent global checkpoints containing the set $S$. Proofs of these facts can be found in [13]. Wang [23] discusses the applications of maximal and minimal consistent global checkpoints in detail.

## 3 A Characterization of Quasi-Synchronous Checkpointing

As we saw earlier (Fig. 1), when processes take checkpoints asynchronously, some or all of the checkpoints taken may be useless. In quasi-synchronous checkpointing, processes take communication-induced checkpoints to reduce the number of useless checkpoints; the message pattern and knowledge gained about the dependency between checkpoints of processes trigger communication-induced checkpoints so that the number of useless checkpoints is minimized or eliminated. Let us first understand how checkpoints become useless and how we can convert useless checkpoints into useful checkpoints.

**Definition 5.** *A* noncausal Z-path *from a checkpoint $C_{p,i}$ to a checkpoint $C_{q,j}$ is a sequence of messages*

$$m_1, m_2, \cdots, m_n \ (n \geq 2)$$

*satisfying the conditions of Definition 2 such that for at least one $i$ $(1 \leq i < n)$, $m_i$ is received by some process $P_r$ after sending the message $m_{i+1}$ in the same checkpoint interval.*

Thus, noncausal Z-paths are those Z-paths that are not causal paths; in particular, Z-cycles are noncausal Z-paths. By Theorem 1, if there exists a noncausal Z-path between two (not necessarily distinct) checkpoints, then the two checkpoints together are useless for constructing a consistent global checkpoint. Moreover, noncausal Z-paths between checkpoints are hard to track on-line and, hence,

3. A causal path from a checkpoint *A* to a checkpoint *B* exists if and only if there exists a sequence of messages $m_1, m_2, \cdots, m_n$ such that $m_1$ is sent after $A$, $m_n$ is received before *B* and $m_i$ is received by some process before the same process sends $m_{i+1} (1 \leq i < n)$.

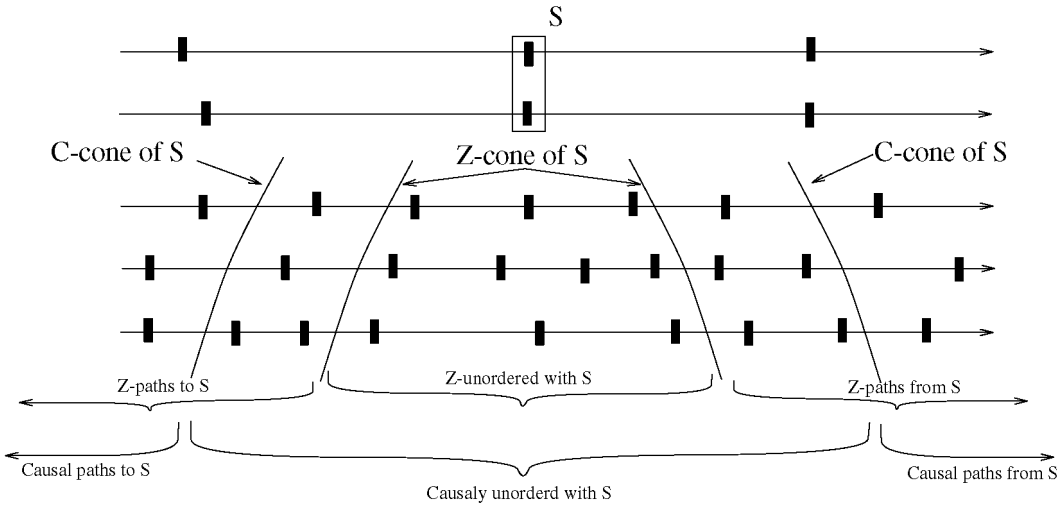


Fig. 2. The *Z-cone* and the *C-cone* of a set of checkpoints $S$ such that $S \not\rightsquigarrow S$.

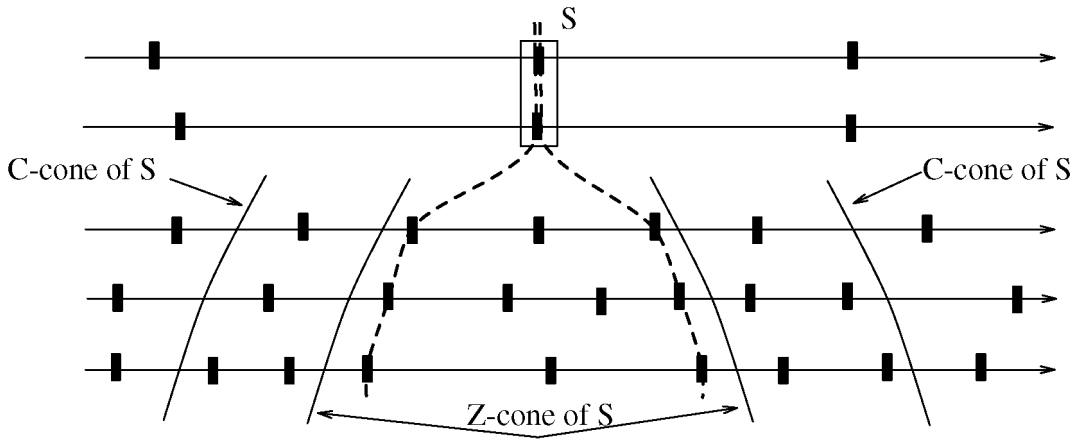


Fig. 3. The minimal and the maximal consistent global checkpoints containing a target set $S$ such that $S \not\rightsquigarrow S$.

the presence of noncausal Z-paths complicates the task of finding consistent global checkpoints. However, noncausal Z-paths between checkpoints are preventable if processes take additional checkpoints at appropriate times. For example, in Fig. 4, the message sequence $m_1, m_2$ constitutes a noncausal Z-path from $C_{1,1}$ to $C_{3,1}$ since $m_2$ is sent before receiving the message $m_1$ in the same checkpoint interval; if $P_2$ took a checkpoint $A$ before receiving the message $m_1$ but after sending the message $m_2$, then this noncausal Z-path could have been prevented and as a result the checkpoints $C_{1,1}$ and $C_{3,1}$ could have been used to construct the consistent global checkpoint $\{C_{1,1}, A, C_{3,1}\}$. Similarly, the message sequence $m_3, m_1$ is a noncausal Z-path from $C_{2,2}$ to itself (in fact, a Z-cycle); this Z-cycle could have been prevented if process $P_1$ took a checkpoint $B$ after sending the message $m_1$, but before receiving message $m_3$, which would have made $C_{2,2}$ useful for constructing a consistent global checkpoint (in fact, $\{B, C_{2,2}, C_{3,1}\}$ would have been one such consistent global checkpoint).

Thus, even though noncausal Z-paths between checkpoints are harmful, they are preventable if processes take additional checkpoints at appropriate places. Preventing all the noncausal Z-paths between checkpoints by making processes take additional checkpoints at appropriate places not only makes all the checkpoints useful but also facilitates construction of consistent global checkpoints incrementally and easily; this is because, in the absence of noncausal Z-paths, any set of checkpoints that are not pairwise causally related can be extended to a consistent global checkpoint by Theorem 1 and causality between checkpoints can be tracked on-line by using vector timestamps [16], [18] or similar mechanisms.

Thus, the primary issues involved in designing a quasi-synchronous checkpointing algorithm are 1) how to efficiently determine appropriate events for processes to take communication induced checkpoints so that noncausal Z-paths can be eliminated and 2) how to minimize the number of communication induced checkpoints taken. Depending upon the strategy adopted to address these issues, noncausal Z-paths between checkpoints can be prevented to varying degrees. Depending on the degree to which the noncausal Z-paths are prevented, quasi-synchronous checkpointing algorithms exhibit different properties and can be classified into various classes. This classification helps to understand the properties and limitations of various checkpointing algorithms, which is helpful for

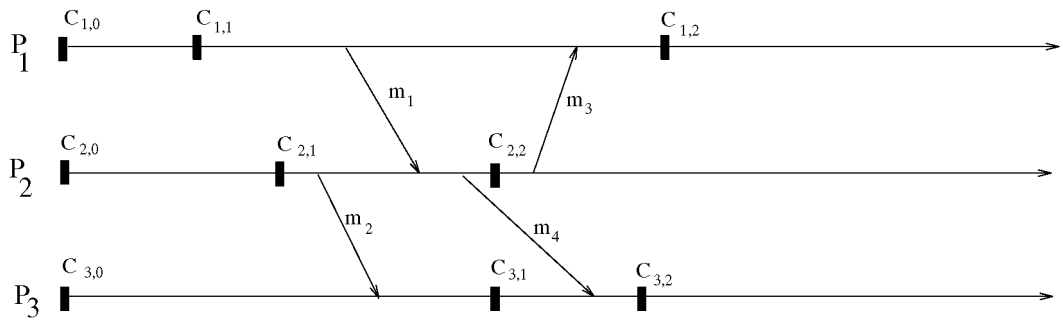


Fig. 4. Noncausal Z-paths.

comparing their performance; it also helps in designing more efficient algorithms. Next, we present a classification of quasi-synchronous checkpointing.

## 4 Classification of Quasi-Synchronous Checkpointing

We classify quasi-synchronous checkpointing algorithms into three different classes, namely, Strictly Z-Path Free (SZPF), Z-Path Free (ZPF), and Z-Cycle Free (ZCF). This classification is based on the degree to which the formation of noncausal Z-paths are prevented. We present the properties of the algorithms belonging to each class and discuss the advantages and disadvantages of algorithms belonging to one class over the other. We also present the relationship of the classification to existing work in the literature.

### 4.1 Strictly Z-Path Free Checkpointing

Strictly Z-path free checkpointing eliminates all the noncausal Z-paths between checkpoints altogether and is the strongest of all the classes. First, we present the definition of a checkpointing pattern that is strictly Z-path free and then discuss the advantages and disadvantages of a system that is strictly Z-path free.

**Definition 6.** *A checkpointing pattern is said to be strictly Z-path free (or SZPF) if there exists no noncausal Z-path between any two (not necessarily distinct) checkpoints.*

In an SZPF system, since there is no noncausal Z-path, a message sequence forms a Z-path if and only if it forms a causal path. The following theorem gives the necessary and sufficient conditions for a system to be SZPF. This theorem is helpful in verifying if a given checkpointing algorithm makes the system SZPF.

**Theorem 2.** *A checkpointing pattern is SZPF if and only if, in every checkpoint interval, all the message-receive events precede all the message-send events.*

**Proof.** ($\Rightarrow$) Suppose there exists a checkpoint interval in which a message-send event precedes a message-receive event. In other words, there exists a process $P_p$ and messages $m_1$ and $m_2$ such that $P_p$ sends $m_2$ and then receives $m_1$ in the same checkpoint interval. Suppose $m_1$ is sent by $P_q$ after taking checkpoint $A$ and $m_2$ is received by $P_r$ before taking checkpoint $B$. Then, the message sequence $m_1, m_2$ forms a noncausal Z-path from $A$ to $B$ since $m_2$ is sent in the same checkpoint interval before receiving $m_1$.

This implies that the system is not SZPF. Hence, in every checkpoint interval of an SZPF system, all the message-receive events precede all the message-send events.

($\Leftarrow$) Conversely, if all message-receive events precede all the message-send events in each checkpoint interval, then clearly there is no noncausal Z-path between checkpoints and, hence, the system is SZPF. □

From Theorem 2, it is clear that the checkpointing pattern shown in Fig. 5 is SZPF. In this figure, the forced checkpoint $C_{2,1}$ is taken to prevent the noncausal Z-path $m_2, m_5$ from $C_{1,0}$ to $C_{3,1}$. Similarly, the forced checkpoint $C_{2,3}$ is taken to prevent the noncausal Z-path $m_3, m_6$ from $C_{1,1}$ to $C_{3,2}$; the forced checkpoint $C_{1,2}$ is taken to prevent the noncausal Z-path $m_4, m_3$ (in fact a Z-cycle) from $C_{2,4}$ to itself. We next discuss the properties of an SZPF system.

#### 4.1.1 Properties of an SZPF System

An SZPF system has many interesting and desirable properties. Since an SZPF system does not allow noncausal Z-paths, Z-cycles do not exist in an SZPF system. Hence, each checkpoint is useful by Corollary 1. Moreover, absence of noncausal Z-paths makes construction of consistent global checkpoints easy because we need to worry about causal paths only and causal paths are easy to track using vector timestamps or other similar mechanisms.

In an SZPF system, any pair of checkpoints between which there is no causal path is useful for constructing a consistent global checkpoint. In fact, a more general result is stated in the following theorem. The following theorem not only presents a necessary and sufficient condition for a given set of local checkpoints to be part of a consistent global checkpoint but also provides a method for constructing them incrementally.

**Theorem 3.** *In an SZPF system, a set of checkpoints $S$ can be extended to a consistent global checkpoint iff $S \not\rightsquigarrow S$.*

**Proof.** In an SZPF system, for any two checkpoints $A$ and $B$, $A \overset{cp}{\rightsquigarrow} B$ if and only if $A \overset{zp}{\rightsquigarrow} B$. Hence, for any set of checkpoints $S$, $S \overset{cp}{\not\rightsquigarrow} S$ if and only if $S \overset{zp}{\not\rightsquigarrow} S$. Hence, the proof follows from Theorem 1. □

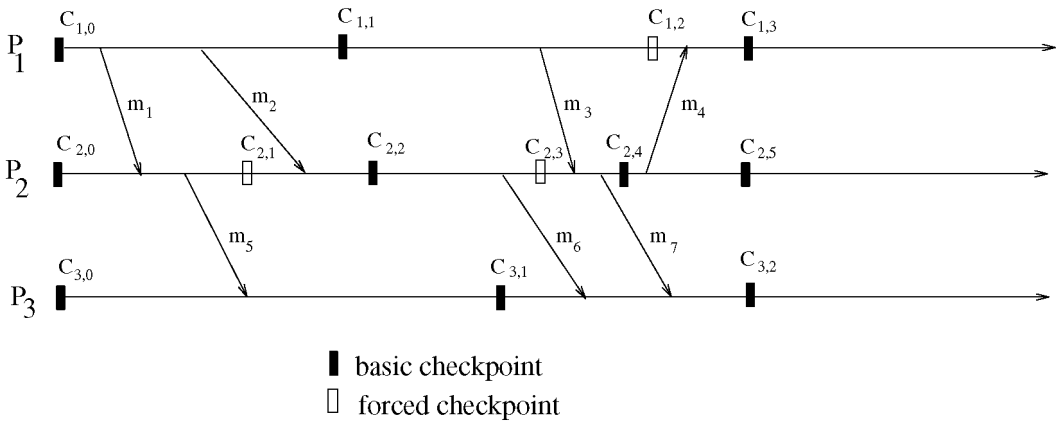


Fig. 5. An SZPF checkpointing.

Thus, in an SZPF system, if $S$ is any set of checkpoints that are not pairwise causally related (i.e., $S \not\rightsquigarrow S$), each process not represented in $S$ is guaranteed to have a checkpoint $A$ such that $A \not\rightsquigarrow S$ and $S \not\rightsquigarrow A$. After adding such a checkpoint $A$ to $S$, the resulting set $S' = S \cup \{A\}$ has the property $S' \not\rightsquigarrow S'$. Thus, we can incrementally extend $S$ to a consistent global checkpoint. Since causality (i.e., $\rightsquigarrow$ relation) can be tracked on-line using vector timestamps or similar other mechanisms [16], [18], constructing consistent global checkpoints incrementally using this method is simple and practical. In a non-SZPF system, however, it is not easy to construct consistent global checkpoints incrementally due to the presence of noncausal Z-paths because tracking noncausal Z-paths on-line is difficult.

In an SZPF system, every Z-path is a causal path and, hence, Z-cone($S$) and C-cone($S$) are identical for any given set of checkpoints $S$. The fact that the Z-cone($S$) and the C-cone($S$) are identical in an SZPF system facilitates the construction of the maximal and the minimal consistent global checkpoints containing a given set $S$. If $S \not\rightsquigarrow S$, then by adding to $S$ the latest checkpoint from each process that is not causally related to any of the checkpoints in $S$ (i.e., the checkpoints lying on the trailing edge of the C-cone($S$)), we can obtain the maximal consistent global checkpoint containing $S$. The minimal consistent global checkpoint containing $S$ can be constructed by adding to $S$ the earliest checkpoint from each process that is not causally related to any of the checkpoints in $S$ (i.e., the checkpoints lying on the leading edge of the C-cone($S$)).

#### 4.1.2 Relation to Existing Work

**The MRS protocol:** The MRS protocol of Russel [19] (also called No-Receive-After-Send (NRAS) by Wang [23]), which was also independently proposed by Acharya and Badrinath [1], disallows any message to be received in any checkpoint interval once a message has been sent in that checkpoint interval. Thus, all message send events precede all message receive events in each checkpoint interval. Hence, it follows from Theorem 2 that the MRS protocol makes the system SZPF. A distributed computation taking checkpoints using the MRS protocol is shown in Fig. 6.

**The Checkpoint-After-Send method:** In Checkpoint-After-Send (CAS) method [23], a checkpoint must be taken after every send event. Thus, any checkpoint interval can have at most one message-send event and it must appear at the end of the interval. Hence, CAS method of checkpointing makes the system SZPF by Theorem 2. Since MRS protocol allows several message-send events to take place in the same checkpoint interval, the CAS method will have higher checkpointing overhead than the MRS method. However, the CAS checkpointing method has the following very interesting and useful property: "*The set consisting of all the latest checkpoints of all the processes forms a consistent global checkpoint*." This property, however, comes at the expense of high checkpointing overhead.

**The Checkpoint-Before-Receive method:** In Checkpoint-Before-Receive (CBR) method [23], a checkpoint must be taken before every receive event. Thus, any checkpoint interval can have at most one message-receive event and it always appears at the beginning of the interval. Hence, it makes the system SZPF by Theorem 2. The CAS method and the CBR method will have the same checkpointing overhead since the number of checkpoints taken in both cases is equal to the number of messages. However, in the CBR method, the latest checkpoints of processes do not form a consistent global checkpoint.

**The Checkpoint-After-Send-Before-Receive method:** In the Checkpoint-After-Send-Before-Receive (CASBR) method [23], a checkpoint must be taken after every message-send event and before every message-receive event and, hence, makes the system SZPF by Theorem 2. Clearly, in the CASBR method, processes take twice as many forced checkpoints as in either CAS method or CBR method.

### 4.2 Z-Path Free Checkpointing

In an SZPF system, absence of noncausal Z-paths between checkpoints makes all the checkpoints useful and also facilitates the construction of consistent global checkpoints incrementally. We can have these desirable features of an SZPF system without actually eliminating *all* noncausal Z-paths. It turns out that we can get these benefits by eliminating only those noncausal Z-paths corresponding to

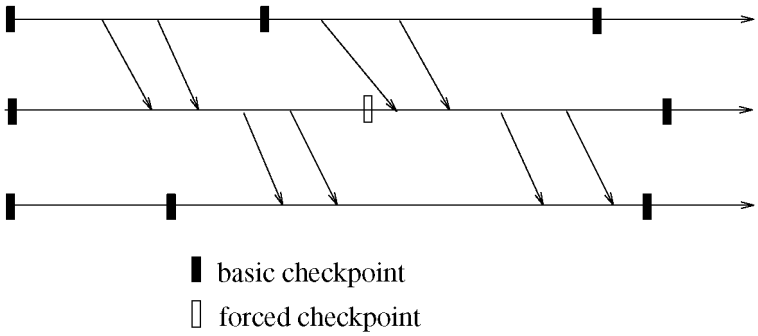


Fig. 6. Checkpointing using MRS protocol.

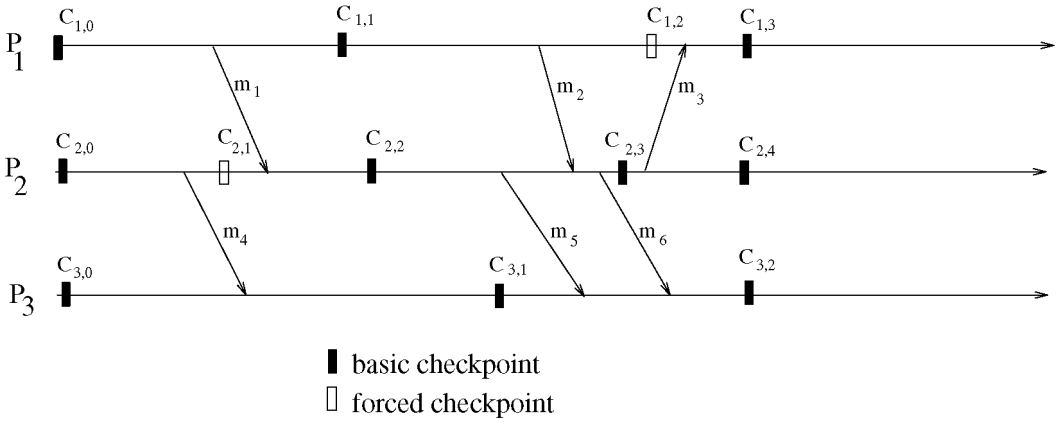


Fig. 7. Checkpointing in ZPF method.

which there is no sibling causal path.[4] This relaxed requirement yields the ZPF model defined below.

**Definition 7.** *A checkpointing pattern is said to be Z-path free (or ZPF) iff for any two checkpoints $A$ and $B$, $A \overset{zp}{\rightsquigarrow} B$ iff $A \overset{cp}{\rightsquigarrow} B$.*

Thus, in a ZPF system, even though noncausal Z-paths may exist between checkpoints, they always have a sibling causal path and, thus, all such noncausal Z-paths can be tracked on-line through the corresponding sibling causal paths. Fig. 7 shows a distributed computation that is ZPF but not SZPF. In this figure, forced checkpoints are taken to prevent noncausal Z-paths corresponding to which there exists no sibling causal path. For example, forced checkpoint $C_{1,2}$ is taken by $P_1$ to prevent noncausal Z-path $m_3, m_2$ (in fact, a Z-cycle from $C_{2,3}$ to itself); forced checkpoint $C_{2,1}$ is taken by process $P_2$ to prevent the Z-path $m_1, m_4$ (a Z-path from $C_{1,0}$ to $C_{3,1}$). However, even though there exists a noncausal Z-path $m_2, m_5$ from $C_{1,1}$ to $C_{3,2}$, no forced checkpoint is taken by process $P_2$ to prevent this Z-path because there exists a sibling causal path $m_2, m_6$.

### 4.2.1 Properties of a ZPF System

A ZPF system has all the interesting properties of an SZPF system. In a ZPF system, Z-cycles do not exist because if a Z-cycle exists then a causal cycle would exist as a sibling; however, causal cycles cannot exist because an event cannot happen before itself. Thus, all checkpoints in a ZPF system are useful. The following theorem gives a necessary and sufficient condition for a given set of local checkpoints to be part of a consistent global checkpoint and also provides a method for constructing them incrementally.

**Theorem 4.** *In a ZPF system, a set of checkpoints $S$ can be extended to a consistent global checkpoint iff $S \overset{cp}{\not\rightsquigarrow} S$.*

**Proof.** In a ZPF system, for any two checkpoints $A$ and $B$, $A \overset{cp}{\rightsquigarrow} B$ if and only if $A \overset{zp}{\rightsquigarrow} B$. Hence, for any set of checkpoints $S$, $S \overset{cp}{\not\rightsquigarrow} S$ if and only if $S \overset{zp}{\not\rightsquigarrow} S$. Hence, the proof follows from Theorem 1. □

The following lemma shows that SZPF $\Longrightarrow$ ZPF.

**Lemma 1.** *If a system is SZPF, then it is ZPF, but the converse is not true.*

**Proof.** In an SZPF system, there is no noncausal Z-path between any two (not necessarily distinct) checkpoints, which trivially implies that for any two checkpoints $A$ and $B$, $A \overset{zp}{\rightsquigarrow} B$ iff $A \overset{cp}{\rightsquigarrow} B$. Hence, an SZPF system is a ZPF system. The converse is not true. For example, the checkpointing pattern in Fig. 7 is ZPF but not SZPF, since $m_2, m_5$ is a noncausal Z-path. □

It is also easy to see that for any given set of checkpoints $S$, the Z-cone($S$) and the C-cone($S$) are identical in a ZPF system and, hence, finding the maximal and minimal consistent global checkpoints containing a target set of

4. If there exists a Z-path from $A$ to $B$ and also a causal path from $A$ to $B$, the causal path is called a sibling of the Z-path.

checkpoints is simple. Thus, a ZPF system has *all* the important features of an SZPF system—constructing consistent global checkpoints incrementally is simple and every checkpoint taken is useful for constructing consistent global checkpoint. In addition, for a given computation, ZPF checkpointing is likely to have less checkpointing overhead than any SZPF checkpointing. This is because in ZPF checkpointing, processes have to take forced checkpoints *only* to prevent noncausal Z-paths corresponding to which there exists no causal path, whereas in SZPF checkpointing, processes have to take forced checkpoints to prevent *all* the noncausal Z-paths.

#### 4.2.2 Equivalence of RD-Trackable and ZPF Systems

We show that the Rollback Dependency Trackable System (RD-Trackable System) of Wang [23] is equivalent to the ZPF system. We define the RD-Trackable system using the terminology of Z-paths; this definition is equivalent to the original definition of [23]. Each process $P_p$ maintains a vector $D_p$ of size $N$. Entry $D_p[p]$, initialized to 1, is incremented every time a new checkpoint is taken and, thus, always represents the current interval number or equivalently, the sequence number of the next checkpoint of $P_p$; every other entry $D_p[q], q \neq p$, is initialized to 0 and records the highest sequence number of any intervals of $P_q$ on which $P_p$'s current state transitively depends. When $P_p$ sends a message $M$, the current value of $D_p$ is piggybacked on $M$. When the receiver $P_q$ receives $M$, $P_q$ updates its vector $D_q$ as follows:

$$D_q[r] := max(M.D[r], D_q[r]), 1 \leq r \leq N,$$

where $M.D$ denotes the vector piggybacked on $M$. When $P_q$ takes the next checkpoint $C_{q,j}$, the value of the vector $D_q$ at that instant is associated with the checkpoint $C_{q,j}$ and is denoted by $C_{q,j}.D$; after taking the checkpoint, $D_q[q]$ is incremented.

**Definition 8.** *A checkpointing pattern is said to satisfy* rollback-dependency trackability *(or is* RD-trackable*) iff for any two checkpoints $C_{p,i}$ and $C_{q,j}$, $C_{p,i} \overset{zp}{\rightsquigarrow} C_{q,j}$ if and only if $C_{q,j}.D[p] \geq i + 1$.*

The following theorem establishes the equivalence of the ZPF system and the RD-trackable system.

**Theorem 5.** *A checkpointing pattern is ZPF if and only if it is RD-trackable.*

**Proof.** From the definition, it follows that in an RD-trackable system, for any two checkpoints $C_{p,i}$ and $C_{q,j}$, $C_{p,i} \overset{zp}{\rightsquigarrow} C_{q,j}$ if and only if $P_q$ received a message $M$ before taking the checkpoint $C_{q,j}$ and $M$ causally depended on a message sent by $P_p$ in its $(i+1)$th checkpoint interval or later (i.e., after taking the checkpoint $C_{p,i}$); in other words, $C_{p,i} \overset{zp}{\rightsquigarrow} C_{q,j}$ if and only if a message $M$ that causally depended on a message sent by $P_p$ after its checkpoint $C_{p,i}$ was received by $P_q$ before the checkpoint $C_{q,j}$. Thus, a system is RD-trackable if and only if, for any two checkpoints $C_{p,i}$ and $C_{q,j}$, $C_{p,i} \overset{zp}{\rightsquigarrow} C_{q,j} \Longleftrightarrow C_{p,i} \overset{cp}{\rightsquigarrow} C_{q,j}$, thus proving the theorem. □

#### 4.2.3 Relation to Existing Work

**Fixed-Dependency-After-Send method:** In Fixed-Dependency-After-Send (FDAS) method [23], when a process $P_p$ sends a message, it piggybacks the current value of the dependency vector $D_p$. After the first message send event in any checkpoint interval of a process $P_q$, if it receives a message $M$, then it processes the message if $M.D[r] \leq D_q[r]\ \forall r$; otherwise, it first takes a checkpoint, updates its dependency vector $D_q$ and then processes the message. Thus, in each checkpoint interval, after the first send event, the dependency vector remains unchanged until the next checkpoint.

**Fixed-Dependency-Interval method:** In Fixed-Dependency-Interval (FDI) method [23], when a process $P_p$ sends a message, it piggybacks the current value of the dependency vector $D_p$. When a process $P_q$ receives a message $M$, $P_q$ processes the message if $M.D[r] \leq D_q[r]\ \forall r$; otherwise, it first takes a checkpoint, updates its dependency vector $D_q$ and then processes the message. Thus, a process is allowed to send and receive messages in a checkpoint interval as long as it will not change the dependency vector in that interval. Venkatesh et al. [22] proposed a checkpointing method that is very similar to the FDI method.

**Baldoni et al.'s method:** Baldoni et al. [2] introduced the notion of causal doubling which is same as the notion of causal sibling that we introduced. They [3] also presented a quasi-synchronous checkpointing algorithm that makes the system ZPF.

Wang [23] showed that both the FDAS method and the FDI method are RD-trackable. Hence, from Theorem 5, both FDAS and FDI methods make the system ZPF. The FDAS, and FDI methods are not SZPF by Theorem 2.

### 4.3 Z-Cycle Free Checkpointing

All checkpoints taken in a ZPF system and an SZPF system are useful. If the objective of a quasi-synchronous checkpointing algorithm is just to make all checkpoints useful, it is not necessary to make the system either ZPF or SZPF. To make all checkpoints useful, it is sufficient to prevent only Z-cycles, from Corollary 1. Thus, we propose a further weaker model below where only Z-cycles are prevented.

**Definition 9.** *A checkpointing pattern is said to be Z-cycle free (or ZCF) iff none of the checkpoints lies on a Z-cycle.*

Fig. 8 shows a distributed computation that is ZCF but not ZPF. In this figure, the forced checkpoint $C_{1,2}$ is taken by $P_1$ to prevent the Z-cycle $m_3, m_2$ from $C_{2,3}$ to itself. The message sequence $m_1, m_4$ forms a noncausal Z-path from $C_{1,0}$ to $C_{3,1}$; also, the message sequence $m_2, m_5$ forms a noncausal Z-path from $C_{1,1}$ to $C_{3,2}$; however, $P_2$ does not take forced checkpoints to prevent these noncausal Z-paths.

The following theorem gives a sufficient condition for a system to be ZCF. For any message $M$, let $M.sn$ denote the sequence number of the latest checkpoint of the sender of $M$ that precedes the event $send(M)$.

**Theorem 6.** *A checkpointing pattern is ZCF if it satisfies the following condition: For any two checkpoints $C_{p,i}$ and $C_{q,j}$ and a message $M$ sent by $P_p$ and received by $P_q$,*

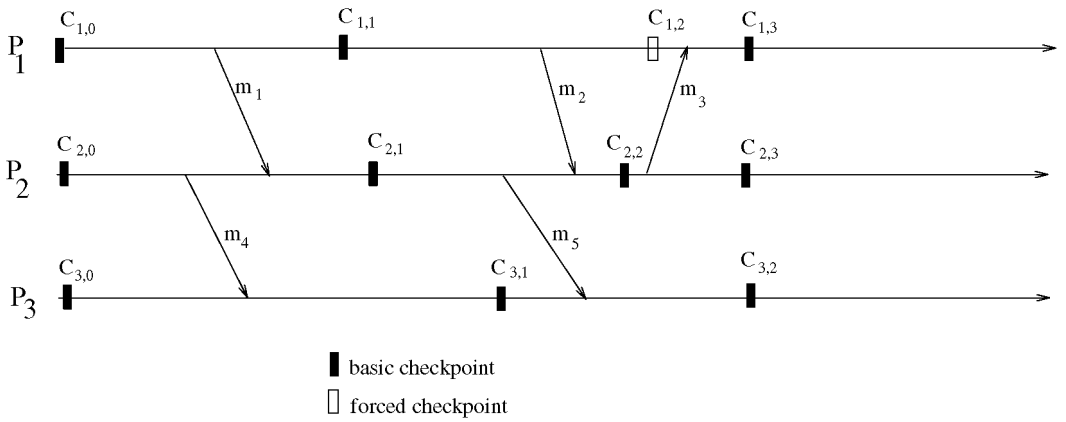


Fig. 8. Checkpointing in ZCF model.

$$M.sn \geq i \Longrightarrow \exists\ j \geq i \text{ such that } (C_{q,j} \xrightarrow{\text{hb}} receive(M)).$$

*(That is, a message sent after taking a checkpoint with sequence number $i$ is received by a process only after taking a checkpoint with sequence number $\geq i$.)*

**Proof.** The proof is by contradiction. Suppose there exists a Z-cycle from checkpoint $C_{p,i}$ to itself. Then, there exists a message sequence $M_1, M_2, \cdots, M_n (n > 1)$ such that

1. $M_1$ is sent by process $P_p$ after $C_{p,i}$,
2. 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 later checkpoint interval (although $M_{k+1}$ may be sent before or after $M_k$ is received), and
3. $M_n$ is received by $P_p$ before $C_{p,i}$ (i.e., $receive(M_n) \xrightarrow{\text{hb}} C_{p,i}$).

Note that for each $k$ $(1 \leq k < n)$, message $M_{k+1}$ is sent in the same or later checkpoint interval in which $M_k$ is received. Since $M_1.sn \geq i$, it follows from the condition given in the theorem and condition 2 above that $M_k.sn \geq i\ \forall k\ (1 \leq k \leq n)$. In particular, $M_n.sn \geq i$. Hence, $\exists\ j \geq i$ such that $C_{p,j} \xrightarrow{\text{hb}} receive(M_n)$, which is a contradiction to the fact that $receive(M_n) \xrightarrow{\text{hb}} C_{p,i}$. Hence, no checkpoint lies on a Z-cycle. Hence, the theorem. □

Theorem 6 could be useful in verifying whether a given checkpointing algorithm makes the system ZCF. However, note that the condition given in the theorem is only sufficient but not necessary for a system to be ZCF.

#### 4.3.1 Properties of a ZCF System

An important feature of a ZCF system is that every checkpoint taken is useful since none of the checkpoints is on a Z-cycle. It is also easy to see that a ZPF system is a ZCF system but not conversely. A ZCF system allows the formation of noncausal Z-paths among checkpoints that are not Z-cycles; therefore, it has less checkpointing overhead than a ZPF system. If a Z-path between two checkpoints is not prevented, the two checkpoints together cannot be part of a consistent global checkpoint; however, individually the two checkpoints can still be part of a consistent global checkpoint if they are not on Z-cycles. For example, in Fig. 8, the checkpoints $C_{1,1}$ and $C_{3,2}$ cannot be part of a consistent global checkpoint together because of the Z-path $m_2, m_5$; however, the sets $\{C_{1,1}, C_{2,1}, C_{3,1}\}$ and $\{C_{1,2}, C_{2,2}, C_{3,2}\}$ are consistent global checkpoints containing $C_{1,1}$ and $C_{3,2}$, respectively.

Even though every checkpoint in a ZCF system is useful, constructing a consistent global checkpoint incrementally is difficult due to the presence of noncausal Z-paths, which are difficult to track on-line. There is a trade-off between weaker checkpointing model and the easeness of constructing consistent global checkpoints.

Next, we present some ZCF quasi-synchronous checkpointing algorithms and explain how they handle the problem of finding consistent global checkpoints.

#### 4.3.2 Relation to Existing Work

**Briatico et al.'s algorithm:** The algorithm of Briatico et al. [5] forces the receiver of a message to take a checkpoint if the sender's checkpoint interval number tagged with the message is higher than the current checkpoint interval number of the receiver. From Theorem 6, this checkpointing method makes all checkpoints Z-cycle free because a message sent in a checkpoint interval is never received in a checkpoint interval with a lower interval number. Checkpoints with the same sequence number form a consistent global checkpoint.

**Manivannan and Singhal's Algorithm:** In the checkpointing algorithm of Manivannan and Singhal [15], each process maintains a counter which is periodically incremented. When a process takes a checkpoint, it assigns the current value of its counter as the sequence number for the checkpoint. Each message is piggybacked with the sequence number of the current checkpoint. If the sequence number accompanying the message is greater than the sequence number of the current checkpoint of the process receiving the message, then the receiving process takes a checkpoint and assigns the sequence number received in the message as the sequence number to the new checkpoint and then processes the message. Since a message sent after a checkpoint with sequence number $i$ is never received by

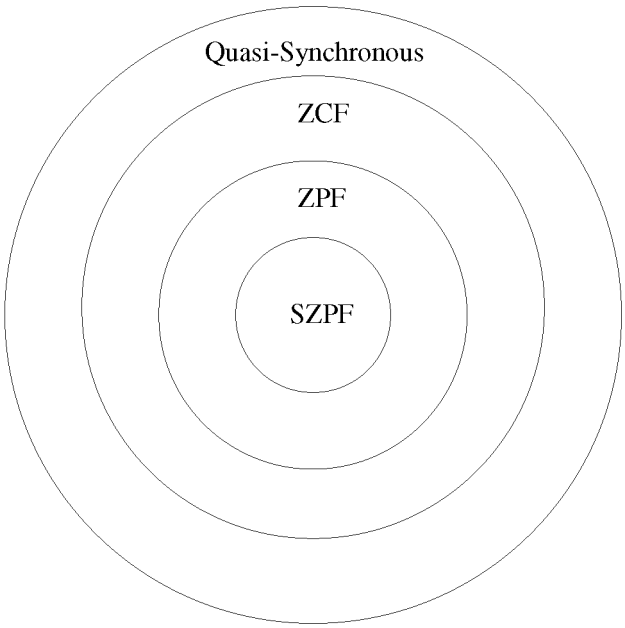


Fig. 9. Relationship between the various checkpointing models proposed in the paper.

any process before taking a checkpoint with sequence number $\geq i$, the system is ZCF by Theorem 6. It is proven in [15] that given a checkpoint $C_{p,i}$, the set

$$S_i^p = \{C_{q,j} \mid j \text{ is the smallest positive integer } \geq \text{i}\}$$

is a consistent global checkpoint containing $C_{p,i}$.

Both the ZCF checkpointing algorithms presented above do not actually track Z-cycles to prevent them. They prevent Z-cycles using a heuristics and as a result they may force processes to take forced checkpoints even when there is no chance for the formation of Z-cycles.

## 5 Discussion

Existing quasi-synchronous checkpointing algorithms in the literature reduce the number of useless checkpoints by preventing the formation of noncausal Z-paths between checkpoints. Based on the extent to which they reduce the useless checkpoints, we classified them into three classes, namely, SZPF, ZPF, and ZCF and also showed that SZPF $\Longrightarrow$ ZPF $\Longrightarrow$ ZCF. Fig. 9 illustrates the relationship among these three classes. We summarize the advantages of the three systems below.

**SZPF system:** Every checkpoint is useful. The absence of noncausal Z-paths makes the construction of consistent global checkpoints easy.

**ZPF system:** Every checkpoint is useful. Existence of a causal sibling for every noncausal Z-path makes the construction of consistent global checkpoints easy. Has the potential to have lower checkpointing overhead than an SZPF system.

**ZCF system:** Every checkpoint is useful. Existence of noncausal Z-paths between checkpoints makes the construction of consistent global checkpoints difficult. Has the potential to have lower checkpointing overhead than a ZPF system.

In terms of finding consistent global checkpoints and making checkpoints useful for the purpose of constructing consistent global checkpoint, ZPF system has the same advantages as an SZPF system. However, an optimal algorithm that makes the system ZPF is better than any algorithm that makes the system SZPF because it has the potential for having less checkpointing overhead. The impossibility of designing an optimal ZPF quasi-synchronous checkpointing algorithm has been addressed by Tsai et al. [20].

In a ZCF system, all the checkpoints are useful. However, due to the presence of noncausal Z-paths between checkpoints, constructing consistent global checkpoints incrementally is difficult. There are no efficient methods for constructing consistent global checkpoints in a ZCF system. So, finding a method to construct consistent global checkpoints efficiently in a ZCF system remains an open problem. The impossibility of designing an optimal ZCF quasi-synchronous checkpointing algorithm has been addressed by Tsai et al. [21].

## 6 Conclusion

When processes take checkpoints independently, some or all of the checkpoints taken may be useless for the purpose of constructing consistent global checkpoints. Quasi-synchronous checkpointing algorithms force processes to take communication induced checkpoints to reduce the number of useless checkpoints. Depending on the extent to which the useless checkpoints are reduced, we classified the quasi-synchronous checkpointing algorithms into various classes. This classification provides a clear understanding of the quasi-synchronous checkpointing algorithms. We also discussed the merits of checkpointing algorithms belonging to one class over the checkpointing algorithms belonging to other classes. This classification also helps in designing more efficient algorithms and evaluating existing algorithms. We pointed out that finding an efficient method to determine consistent global checkpoints in a ZCF system remains an open problem.

## Acknowledgments

The authors thank the anonymous referees for their excellent suggestions for improvement of the paper.

## References

[1] A. Acharya and B.R. Badrinath, "Checkpointing Distributed Applications on Mobile Computer," *Proc. Third Int'l Conf. Parallel and Distributed Information Systems*, Sept. 1994.

[2] 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.

[3] R. Baldoni, J.M. Helary, A. Mostefaoui, and M. Raynal, "A Communication Induced Algorithm that Ensures the Rollback Dependency Trackability," *Proc. 27th Int'l Symp. Fault-Tolerant Computing*, Seattle, Wash., July 1997.

[4] B. Bhargava and S.R. Lian, "Independent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems—An Optimistic Approach," *Proc. Seventh IEEE Symp. Reliable Distributed Systems,* pp. 3-12, 1988.
[5] D. Briatico, A. Ciuffoloetti, and L. Simoncini, "A Distributed Domino-Effect Free Recovery Algorithm," *Proc. IEEE Fourth Symp. Reliability in Distributed Software and Database Systems,* pp. 207-215, 1984.
[6] L. Moura e Silva and J.G. Silva, "Global Checkpointing for Distributed Programs," *Proc. Symp. Reliable Distributed Systems,* pp. 155-162, 1992.
[7] K.H. Kim, "Programmer-Transparent Coordination of Recovering Concurrent Processes: Philosophy and Rules for Efficient Implementation," *IEEE Trans. Software Eng.,* vol. 14, no. 6, pp. 810-821, June 1988.
[8] 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.
[9] 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.
[10] K. Tsuruoka, A. Kaneko, and Y. Nishihara, "Dynamic Recovery Schemes for Distributed Process," *Proc. IEEE Second Symp. Reliability in Distributed Software and Database Systems,* pp. 124-130, 1981.
[11] L. Lamport, "Time, Clocks and Ordering of Events in Distributed Systems," *Comm. ACM,* vol. 21, no. 7, pp. 558-565, July 1978.
[12] K. Li, J.F. Naughton, and J.S. Plank, "Checkpointing Multicomputer Application," *Proc. 10th Symp. Reliable Distributed Systems,* pp. 2-11, 1991.
[13] D. Manivannan, R.H.B. Netzer, and M. Singhal, "Finding Consistent Global Checkpoints in a Distributed Computation," Technical Report OSU-CISRC-3/96-TR16, The Ohio State Univ., Dept. of Computer and Information Science, 1996.
[14] D. Manivannan, R.H.B. Netzer, and M. Singhal, "Finding Consistent Global Checkpoints in a Distributed Computation," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 6, pp. 623-627, June 1997.
[15] D. Manivannan and M. Singhal, "A Low-Overhead Recovery Technique Using Quasi-Synchronous Checkpointing," *Proc. 16th Int'l Conf. Distributed Computing Sytems,* pp. 100-107, Hong Kong, May 1996.
[16] F. Mattern, "Virtual Time and Global States of Distributed Systems," *Parallel and Distributed Algorithms,* M. Cosnard et al., eds., pp. 215-226. North Holland: Elsevier Science, 1989.
[17] 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.
[18] M. Raynal and M. Singhal, "Logical Time: Capturing Causality in Distributed Systems," *Computer,* vol. 29, no. 2, pp. 49-56, Feb. 1996.
[19] D. Russel, "State Restoration in Systems of Communicating Processes," *IEEE Trans. Software Eng.,* vol. 6, no. 2, pp. 183-194, 1980.
[20] J. Tsai, S.-Y. Kuo, and Y.-M. Wang, "Theorectical Analysis for Communication-Induced Checkpointing Protocols with Rollback-Dependency Trackability," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 10, pp. 963-971, Oct. 1998.
[21] J. Tsai, Y.-M. Wang, and S.-Y. Kuo, "Evaluation of Domino-Free Communication-Induced Checkpointing Protocols," Research Report MSR-TR-98-43, Microsoft Corp., http://www.research.microsoft.com/scripts/pubdb/trpub.asp, Sept. 1998.
[22] K. Venkatesh, T. Radhakrishnan, and H.F. Li, "Optimal Checkpointing and Local Encoding for Domino-Free Rollback Recovery," *Information Processing Letters,* vol. 25, pp. 295-303, July 1987.
[23] 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.
[24] J. Xu and R.H.B. Netzer, "Adaptive Independent Checkpointing for Reducing Rollback Propagation," *Proc. Fifth IEEE Symp. Parallel and Distributed Processing,* Dec. 1993.

**D. Manivannan** received a BSc degree in mathematics with special distinction 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, in 1992 and 1993, respectively. He received his PhD degree in computer science from The Ohio State University in 1997.

Dr. Manivannan is currently an assistant professor of computer science at the University of Kentucky, Lexington. Dr. Manivannan's research interests include distributed systems, operating systems, mobile computing systems, and interprocess communication in parallel architectures. Dr. Manivannan is a member of the ACM, the IEEE, and the IEEE Computer Society.

**Mukesh Singhal** received a Bachelor of Engineering degree in electronics and communication engineering with high distinction from the University of 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, computer security, and performance modeling. He has published more than 100 refereed articles in these areas. He has coauthored two books titled "Advanced Concepts in Operating Systems" (McGraw-Hill, 1994), and "Readings in Distributed Computing Systems" (IEEE CS Press, 1993). He is currently the program director of the Operating Systems and Compilers Program at the U.S. National Science Foundation. He is a senior member of the IEEE.