On the "No-Z-Cycle" Property in Distributed Executions[*]

Francesco QUAGLIA Roberto BALDONI Bruno CICIANI

Dipartimento di Informatica e Sistemistica, Università "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
E.mail: {quaglia,baldoni,ciciani}@dis.uniroma1.it

Abstract

Given a checkpoint and communication pattern of a distributed execution, the "No Z-Cycle" property ($\mathcal{NZC}$) states that there does not exist a dependency between a checkpoint and itself. In other words, there does not exist a non-causal sequence of messages that starts after a checkpoint and terminates before that checkpoint. From an operational point of view, this property corresponds to the fact that each checkpoint belongs to at least one consistent global checkpoint. So it could be used, for example, for restarting a distributed application after the occurrence of a failure. In this paper we derive a characterization of the $\mathcal{NZC}$ property (previously an open problem). It identifies a subset of Z-cycles, namely *Core Z-Cycles* (*CZCs*), that has to be empty in order that the checkpoint and communication pattern of the execution satisfies the $\mathcal{NZC}$ property. Then, we present a communication-induced checkpointing protocol that prevents CZCs on-the-fly. This protocol actually removes from the execution checkpoint and communication patterns whose structure is *the common causal part* to any CZC. Finally we propose a taxonomy of communication-induced checkpointing protocols that ensure the $\mathcal{NZC}$ property.

Key terms: checkpointing, causality, reliability, theory of distributed computing, consistent global states, distributed algorithms, distributed systems.

[*]This article appeared in Journal of Computer and System Sciences, vol.61, no.3, December 2000.

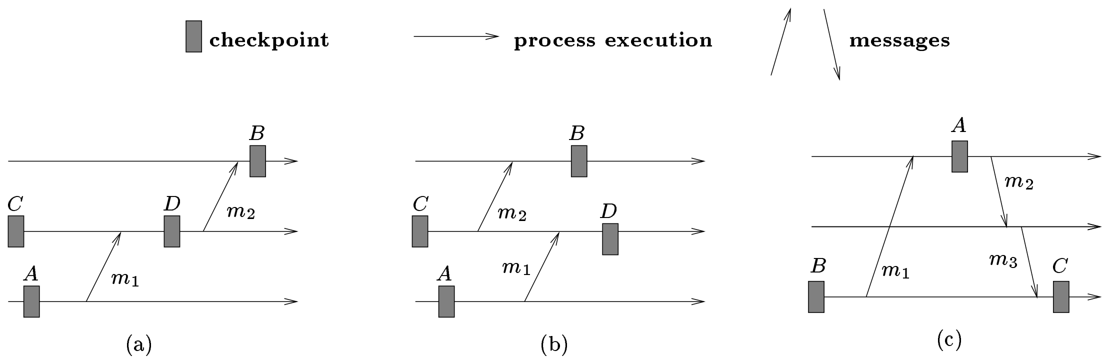


Figure 1: a causal Z-path from A to B (a), a non-causal Z-path from A to B (b), a Z-cycle involving A (c).

1 Introduction

A *checkpoint and communication pattern* of a distributed execution consists of a set of local checkpoints and a dependency relation defined over them due to interprocess communication [23]. A *local checkpoint* is the local state of a process recorded on stable storage. Dependencies are caused by chains of messages in the execution called *zigzag paths* (*Z-paths* for short) [18]. A Z-path is a particular sequence of messages $[m_1, \ldots, m_q]$ such that the sending of a message m_i belongs to the same, or to a successive, checkpoint interval of the delivery of the message m_{i-1} (a *checkpoint interval* is the set of events between two successive checkpoints in the same process). Z-paths can be split in two families: *causal Z-paths* which are actually causal paths (as an example a causal Z-path from the checkpoint A to the checkpoint B formed by $[m_1, m_2]$ is shown in Figure 1.a) and *non-causal Z-paths* in which there exists at least one message m_i whose send precedes the delivery of m_{i-1} in the same checkpoint interval (Figure 1.b shows the non-causal Z-path from A to B formed by $[m_1, m_2]$).

Due to the presence of non-causal Z-paths, a message chain could start after a checkpoint and terminate before that checkpoint. In that case a dependency relation is established between a checkpoint and itself. This pattern has been formalized by Netzer and Xu with the name of *Z-cycle* [18]. As an example, the sequence of messages $[m_2, m_3, m_1]$ shown in Figure 1.c involves the checkpoint A in a Z-cycle.

A checkpoint and communication pattern of a distributed execution satisfies the *No-Z-Cycle* property ($\mathcal{NZC}$) if it does not contain any Z-cycle. $\mathcal{NZC}$ has a noteworthy feature: *"a checkpoint which is not involved in any Z-cycle belongs at least to one consistent global checkpoint"*. A global checkpoint of a distributed execution is a collection of local checkpoints, one for each process. A global checkpoint is consistent [5] if, for any pair of its checkpoints, no one *happens-before* ([14]) another. As a consequence the $\mathcal{NZC}$ property has applications in many dependability problems such as determination of *distributed breakpoints* [9], determination of *shared global states* of a distributed execution [10] and *rollback-recovery* [7], just to name the most important. As an example, in the context of rollback-recovery, ensuring the $\mathcal{NZC}$ property means rollback without the risk of *domino effect* [19][1] and simple and efficient solutions for the *recovery line computation* and the *garbage collection problem* [7][2].

As first contribution, this paper presents a characterization of the $\mathcal{NZC}$ property. This result has been obtained thanks to the introduction of concatenation relations on message chains and checkpoints that allow to express, in an easy way, the basic structure of checkpoint and communication patterns. The characterization lies on *Core Z-Cycles* (CZCs) that are Z-cycles with several constraints on their structure. We prove that $\mathcal{NZC} \Leftrightarrow \mathcal{NCZC}$ where the $\mathcal{NCZC}$ property means that the checkpoint and communication pattern of the execution does not contain any CZC ([3]).

The previous characterization is important not only from a theoretical point of view but, also, from a practical one as we can derive checkpointing protocols that ensure on-the-fly the $\mathcal{NZC}$ property. In particular we introduce a family of communication-induced checkpointing protocols ($\mathcal{F}_{\mathcal{NZC}}$) based on the following basic hypothesis: (1) the usable knowledge at a certain event can not be more than the one included in the *causal past* of that event, and (2) the execution is asynchronous i.e., no bound exists on the process' speed and on the message transfer delay. Basing on these hypothesis Z-cycles cannot be tracked on-the-fly as they are non-causal in nature. A protocol preventing the formation of a particular checkpoint and communication pattern, namely *Suspect Core Z-Cycle* (SCZC), is proposed. The SCZC pattern represents the causal part of *any* core Z-cycle. As it is causal, it is on-the-fly trackable by a protocol belonging to $\mathcal{F}_{\mathcal{NZC}}$. The prevention of SCZC patterns is done by taking *forced* checkpoints. This action is directed by a predicate that exploits the control information piggybacked on the incoming messages and the local history of a process. More specifically, we assume each process P_i selects some local states to be local checkpoints (*basic* checkpoints) then, upon the delivery of a message m, a process P_i is directed to take a forced checkpoint if message m is "closing" an SCZC pattern. The proposed protocol piggybacks on each application message a matrix of $n \times n$ integers where n represents the number of processes of the execution.

Recently Helary et al. [11] have shown that ensuring $\mathcal{NZC}$ is a particular application of the *Virtual-Precedence* property ($\mathcal{VP}$), defined on an interval based abstraction of a distributed execution. If the abstraction satisfies $\mathcal{VP}$, then it is possible to associate a *timestamping function* with

[1]The domino effect consists of an unbounded rollback propagation that, upon the occurrence of a failure, forces a distributed application to be resumed from its initial state as no recent consistent global checkpoint can be built while rolling back.

[2]Recovery line computation consists of finding the consistent global checkpoint closest to the end of the execution from which a distributed application can be resumed after a failure. Garbage collection allows to discard from stable storage all checkpoints related to events that occurred before the last computed recovery line.

[3]In the remainder of the paper we denote in uppercase a *specific* checkpoint and communication pattern, in bold uppercase a *set* of checkpoint and communication patterns of the same type and in calligraphic style *properties* related to checkpoint and communication patterns.

intervals such that: intervals which are connected by a message are timestamped in a non-decreasing way (safety part) and the timestamp of a process increases after communication (liveness part). In the checkpointing problem, intervals of the abstraction correspond to checkpoint intervals and the abstraction of the distributed execution corresponds to a checkpoint and communication pattern. In this context it has been shown that $\mathcal{VP} \Leftrightarrow \mathcal{NZC}$ [11]. From the previous equivalence, it comes out that one can ensure $\mathcal{NZC}$ either by using a checkpointing protocol relying on a particular timestamping function that satisfies $\mathcal{VP}$ or by preventing the formation of particular checkpoint and communication patterns which are necessary to form a Z-cycle. Informally, the former method ensures $\mathcal{NZC}$ "passing through" $\mathcal{VP}$ while the second ensures $\mathcal{VP}$ "passing through" $\mathcal{NZC}$.

Basing on previous observation, we propose, as last contribution of this paper, a taxonomy of communication-induced checkpointing protocols in $\mathcal{F}_{\mathcal{NZC}}$ that splits them into two classes according to the way the "see" the equivalence between $\mathcal{VP}$ and $\mathcal{NZC}$, namely $\mathcal{VP}$-enforced and $\mathcal{VP}$-accordant protocols. A $\mathcal{VP}$-enforced protocol assumes the existence of a timestamping function that labels checkpoint intervals. A forced checkpoint is taken each time either safety or liveness of the function is going to be violated. A $\mathcal{VP}$-accordant protocol does not use a timestamping function but prevents the formation of particular checkpoint and communication patterns in the execution. The protocol proposed in this paper belongs to latter class. To the best of our knowledge this is the first taxonomy that splits communication-induced protocols according to a homogeneous criterion, namely, the equivalence between $\mathcal{VP}$ and $\mathcal{NZC}$.

The remainder of the paper is structured as follows. Section 2 introduces the model of the distributed execution. The notion of message chain and the concept of Z-cycle are formally introduced in Section 3. In the same section we introduce two concatenation relations that will be used to express the structure of checkpoint and communication patterns. In Section 4 the CZC is introduced, and the result $\mathcal{NZC} \Leftrightarrow \mathcal{NCZC}$ is proved. Section 5 presents the communication-induced checkpointing protocol derived from the previous characterization. In the final section we discuss the relation between the $\mathcal{NZC}$ property and the *Virtual Precedence* ($\mathcal{VP}$) property introduced by Helary et al. in [11], and we present the taxonomy of checkpointing protocols in $\mathcal{F}_{\mathcal{NZC}}$.

2 The Model of a Distributed Execution

We assume a distributed execution consisting of a set of n processes $\{P_1, P_2, \ldots, P_n\}$. Processes do not share memory and do not share a common clock value. They communicate only by exchanging messages. Each pair of processes is connected by an asynchronous, directed logical channel. Transmission delays over channels are unpredictable but finite.

A process produces a sequence of *events*; each event moves the process from one *local* state to another. The h-th event in process P_i is denoted as $e_{i,h}$. We assume events are produced by the execution of internal, send or deliver statements. The send and deliver events of a message m are denoted respectively by $send(m)$ and $deliver(m)$.

Definition 2.1
In process P_i an event $e_{i,h}$ precedes an event $e_{i,k}$, denoted $e_{i,h} \prec_P e_{i,k}$, if, and only if, $h \leq k$.

Definition 2.2
An event $e_{i,h}$ of process P_i precedes an event $e_{j,k}$ of process P_j due to message m, denoted $e_{i,h} \prec_m e_{j,k}$, if, and only if:

$$(e_{i,h} = \mathit{send}(m)) \wedge (e_{j,k} = \mathit{deliver}(m))$$

Lamport's *Happened-Before* relation [14], denoted as $\overset{e}{\rightarrow}$, is the transitive closure of the union of relations $\prec_P$ and $\prec_m$. Let $\mathcal{H}$ be the set of all the events produced by a distributed execution, the execution can be modeled as a partial order $\widehat{\mathcal{H}} = (\mathcal{H}, \overset{e}{\rightarrow})$.

A local state of a process saved on stable storage is called a *local checkpoint* of the process. A local state is not necessarily recorded as a local checkpoint, so the set of local checkpoints is a subset of the set of local states.

The x-th checkpoint of process P_i is denoted as $C_{i,x}$ where x is called the *rank* of the checkpoint. The rank of checkpoints of a process increases monotonically: each time a checkpoint is taken the rank is increased by one. We assume that each process P_i takes an initial checkpoint $C_{i,1}$ (corresponding to the initial state of the process) and that after each event a checkpoint will eventually be taken. A checkpoint interval $I_{i,x}$ is the set of events between $C_{i,x}$ and $C_{i,x+1}$. Among checkpoint intervals let us define the following relation based on the Happened-Before relation:

Definition 2.3
A checkpoint interval $I_{i,x}$ precedes a checkpoint interval $I_{j,y}$, denoted $I_{i,x} \overset{I}{\rightarrow} I_{j,y}$, if, and only if:

$$\exists e_{i,x'} \in I_{i,x}, \exists e_{j,y'} \in I_{j,y} \quad : \quad e_{i,x'} \overset{e}{\rightarrow} e_{j,y'}$$

Note that precedence between intervals is not transitive. Let us finally introduce the concept of checkpoint and communication pattern related to a distributed execution:

Definition 2.4
A checkpoint and communication pattern of a distributed execution is a pair $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$ where $\widehat{\mathcal{H}}$ is a distributed execution and $\mathcal{C}_{\widehat{\mathcal{H}}}$ is a set of local checkpoints defined on $\widehat{\mathcal{H}}$.

3 Background Theory and Preliminary Definitions

This section introduces the notions of causal and non-causal message chain and two concatenation relations which are used to express the causal, or non-causal, combination of checkpoints and/or chains of messages. Finally, the concept of Z-cycle is reformulated using the concatenation relations.

3.1 Message Chains

Definition 3.1
A message chain is a sequence of messages $\zeta = [m_1, m_2, \ldots, m_\ell]$ *such that:*

$$\forall k : 1 \leq k \leq \ell - 1 \Rightarrow (deliver(m_k) \in I_{i,x}) \wedge (send(m_{k+1}) \in I_{i,y}) \wedge (x \leq y)$$

As an example, in Figure 1.b we have a message chain formed by messages $[m_1, m_2]$. A particular case of message chain is the *causal* message chain, in which the deliver event of a message always precedes on a process the send event of the successive message of the chain. More formally we have:

Definition 3.2
A message chain $\zeta = [m_1, m_2, \ldots, m_\ell]$ *is causal if:*

$$\forall k : 1 \leq k \leq \ell - 1 \Rightarrow deliver(m_k) \prec_P send(m_{k+1})$$

otherwise, the chain is non-causal.

An example of causal message chain is the one formed by messages $[m_1, m_2]$ in Figure 1.a. A chain with only one message is always causal. For the sake of clarity, the Greek letter μ indicates a causal message chain. Furthermore we denote with $\zeta.first$ (resp. $\zeta.last$) the first (resp. last) message of a message chain ζ.

$|\zeta|$ denotes the number of messages forming the chain ζ (i.e., its size). In particular, $|\zeta| = n$ means that the chain ζ consists of n messages. We use the operator *minus* to denote the removal of a subchain from a chain; for example $\zeta - \zeta.last$ (resp. $\zeta - \zeta.first$) denotes a chain obtained from ζ by removing its last (resp. first) message and $\zeta - \widehat{\zeta}$ denotes a chain obtained by removing the subchain $\widehat{\zeta}$ from ζ, where $\widehat{\zeta}$ can be either the initial or the final part of ζ.

Finally, we denote as $S(\zeta) = \{I_{j_1,z_1}, I_{j_2,z_2}, \ldots, I_{j_\ell,z_\ell}\}$ the sequence of checkpoint intervals traversed by $\zeta = [m_1, m_2, \ldots, m_\ell]$, i.e., $S(\zeta) = \{I_{j_i,z_i} | \forall m_i \in \zeta, send(m_i) \in I_{j_i,z_i}\}$.

3.2 Concatenation Relations

Causal Concatenation Relation. The causal concatenation relation, denoted by the symbol $\circ$, expresses the causal combination of two objects (an object can be either a checkpoint or a message chain). It is defined as follows:

Definition 3.3
An object $\mathtt{a}$ *is causally concatenated to an object* $\mathtt{b}$*, denoted* $\mathtt{a} \circ \mathtt{b}$*, if, and only if:*

$$\begin{aligned}
&((\mathtt{a} \equiv C_{i,x}) \wedge (\mathtt{b} \equiv \zeta) \wedge (\exists v \geq 0 \ : \ send(\zeta.first) \in I_{i,x+v})) \vee \\
&((\mathtt{a} \equiv \zeta) \wedge (\mathtt{b} \equiv C_{i,x}) \wedge (\exists v > 0 \ : \ deliver(\zeta.last) \in I_{i,x-v})) \vee \\
&((\mathtt{a} \equiv \zeta) \wedge (\mathtt{b} \equiv \zeta') \wedge (deliver(\zeta.last) \prec_P send(\zeta'.first))
\end{aligned}$$

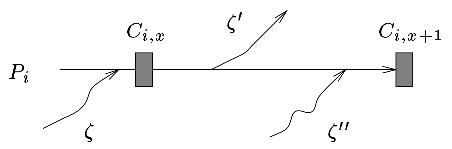


Figure 2: concatenation relations.

The following examples of causal concatenation are shown in Figure 2: (i) $\zeta \circ C_{i,x}$, (ii) $C_{i,x} \circ \zeta'$ and (iii) $\zeta \circ \zeta'$. Note that a Z-path from a checkpoint A to a checkpoint B due to a message chain ζ can be expressed as $A \circ \zeta \circ B$ (for the sake of simplicity $A \circ \zeta \circ B$ stands for $(A \circ \zeta) \wedge (\zeta \circ B)$).

Non-Causal Concatenation Relation. The non-causal concatenation relation, denoted by the symbol $\bullet$, expresses the non-causal combination of message chains. It is defined as follows:

Definition 3.4
A message chain ζ is non-causally concatenated to a message chain ζ' in the checkpoint interval $I_{k,y}$, denoted $\zeta \overset{k,y}{\bullet} \zeta'$, if, and only if:

$$(deliver(\zeta.last) \in I_{k,y}) \wedge (send(\zeta'.first) \in I_{k,y}) \wedge (send(\zeta'.first) \prec_P deliver(\zeta.last))$$

As an example, the non-causal concatenation $\zeta'' \overset{i,x}{\bullet} \zeta'$ is shown in Figure 2. Whenever not necessary the pair of indices is dropped from the non-causal relation.

3.3 Concatenation Operators

Let consider two message chains $\zeta = [m_1, \ldots, m_q]$ and $\zeta' = [m'_1, \ldots, m'_p]$. If $\zeta \circ \zeta'$ (or $\zeta \bullet \zeta'$) then by Definition 3.1, there exists in the checkpoint and communication pattern of the distributed execution a message chain $\bar{\zeta} = [m_1, \ldots, m_q, m'_1, \ldots, m'_p]$. Therefore, whenever two message chains are concatenated (either causally or non-causally), then there exists a chain resulting from that concatenation and containing all messages of the two original chains. This property allows us to use concatenation relations applied to message chains also as concatenation operators generating message chains. For the previous example, the generated message chain is $\bar{\zeta} = \zeta \circ \zeta'$ (or, in the case of non-causal concatenation, $\bar{\zeta} = \zeta \bullet \zeta'$).

3.4 Z-Cycles and the No-Z-Cycle Property

By using previous notations, let us express the notion of *Z-Cycle* (ZC) introduced by Netzer and Xu [18]. Basically, a ZC is a checkpoint and communication pattern involving a checkpoint $C_{i,x}$ and a chain $\widehat{\zeta}$ such that $\widehat{\zeta} \circ C_{i,x} \circ \widehat{\zeta}$ (an example of such a concatenation is shown in Figure 3.a). However, it is always possible to separate $\widehat{\zeta}$ into two sub-chains, a causal sub-chain μ and a sub-chain ζ such that $\widehat{\zeta} = \mu \overset{k,y}{\bullet} \zeta$ (this concatenation is shown in Figure 3.b). This observation gives rise to the following Z-cycle definition:

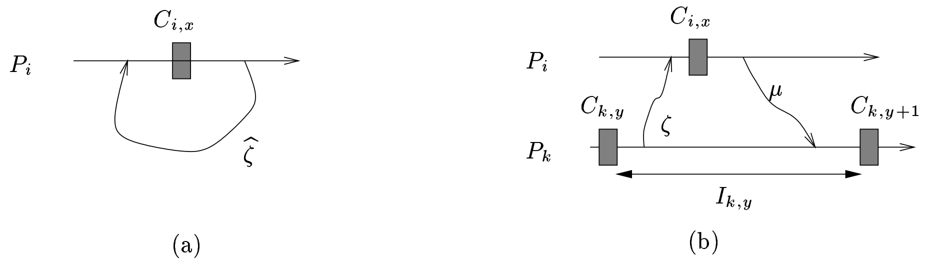


Figure 3: the structure of a Z-Cycle.

Definition 3.5
A ZC is a checkpoint and communication pattern $ZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ *such that:* $\zeta \circ C_{i,x} \circ \mu \overset{k,y}{\bullet} \zeta$.

Let us finally introduce the No-Z-Cycle property:

Property 3.1
A checkpoint and communication pattern of a distributed execution $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$ *satisfies the No-Z-Cycle property* ($\mathcal{NZC}$) *if, and only if, no ZC exists in* $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$.

4 A Characterization of the No-Z-Cycle Property

To get a characterization of the $\mathcal{NZC}$ property, successive embedded subsets of Z-cycles, namely *Prime Z-Cycle* (**PZC**) and *Core Z-Cycle* (**CZC**) are introduced, which contain Z-cycles that satisfy progressively stronger constraints on their checkpoint and communication pattern structure. In particular, a PZC is a $ZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ imposing a constraint on μ and, finally, a CZC is a PZC with a constraint on the sequence of checkpoint intervals associated with ζ.

We prove that: (i) *if* ZC *then* PZC and (ii) *if* PZC *then* CZC. In other words, each (non-core) Z-cycle involving a checkpoint A embeds a core Z-cycle involving a checkpoint B (see Figure 4). This means that **ZC** is empty if, and only if, **CZC** is empty as will be proved in the characterization theorem (Section 4.3).

4.1 Prime Z-Cycles

This paragraph introduces the notion of Prime Z-Cycle (PZC). It is interesting because of the result in Lemma 4.2 which states that if there is a Z-cycle in a checkpoint and communication pattern of a distributed execution then there exists in that checkpoint and communication pattern a PZC whose size of its chain ζ is smaller than, or equal to, the one of the original Z-cycle.

Given a pair $(C_{i,x}, P_k)$, let us consider the set of causal chains μ starting after $C_{i,x}$ whose recipient of $\mu.last$ is P_k denoted $M(C_{i,x}, P_k)$. This set is partially ordered by the relation:

$$\mu \prec \mu' \Leftrightarrow deliver(\mu.last) \prec_P deliver(\mu'.last)$$

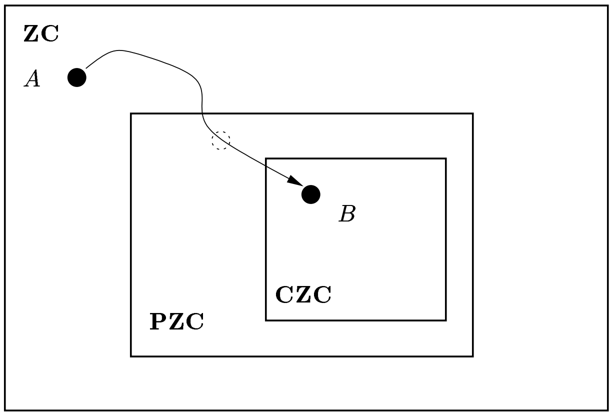


Figure 4: relations between **ZC**, **PZC** and **CZC**.

Let $min(M(C_{i,x}, P_k))$ denote the set of the *minimum* elements in $M(C_{i,x}, P_k)$ ([4]). This set contains causal chains starting after $C_{i,x}$ and sharing the last message. By using these notions the concept of Prime Z-Cycle (PZC) is introduced as follows:

Definition 4.1 *(Prime Z-cycle)*
$ZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ is a PZC, denoted $PZC(C_{i,x}, \mu, I_{k,y}, \zeta)$, if, and only if, $\mu \in min(M(C_{i,x}, P_k))$.

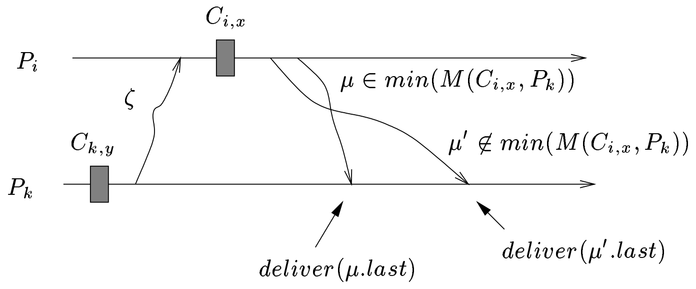


Figure 5: the structure of a ZC and of a PZC.

As an example $ZC(C_{i,x}, \mu', I_{k,y}, \zeta)$ shown in Figure 5 is not a PZC while $ZC(C_{i,x}, \mu, I_{k,y}, \zeta)$, shown in the same figure, is a PZC. Let us introduce the following lemma:

Lemma 4.1
If *there exists $ZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ such that $|\zeta| = 1$*
then *there exists $PZC(C_{i,x}, \mu', I_{k,y}, \zeta)$.*

Proof
Let us consider $ZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ such that $\zeta = m$ (i.e., $|\zeta| = 1$). We have two alternatives:

[4]A chain $\mu \in M(C_{i,x}, P_k)$ is a minimum element if there does not exist any chain $\mu' \in M(C_{i,x}, P_k)$ such that $\mu' \prec \mu$.

1 **if** $\mu \in min(M(C_{i,x}, P_k))$ **then** let us consider $\mu' = \mu$. By Definition 4.1 we get $PZC(C_{i,x}, \mu', I_{k,y}, \zeta)$ and the claim follows;

2 **if** $\mu \notin min(M(C_{i,x}, P_k))$ **then** let us consider $\mu' \in min(M(C_{i,x}, P_k))$ (note that μ' exists as $M(C_{i,x}, P_k)$ is not empty since it contains μ).
There are two cases:

2.1 $deliver(\mu'.last) \overset{e}{\rightarrow} send(m)$ (see Figure 6.a).
This is impossible as it would lead to a cycle in the *Happened-Before* relation (i.e., $send(m) \overset{e}{\rightarrow} deliver(\mu'.last)$) which is acyclic [14];

2.2 $send(m) \overset{e}{\rightarrow} deliver(\mu'.last)$ (see Figure 6.b).
Thus, by Definition 4.1 we get $PZC(C_{i,x}, \mu', I_{k,y}, \zeta)$ and the claim follows.

□

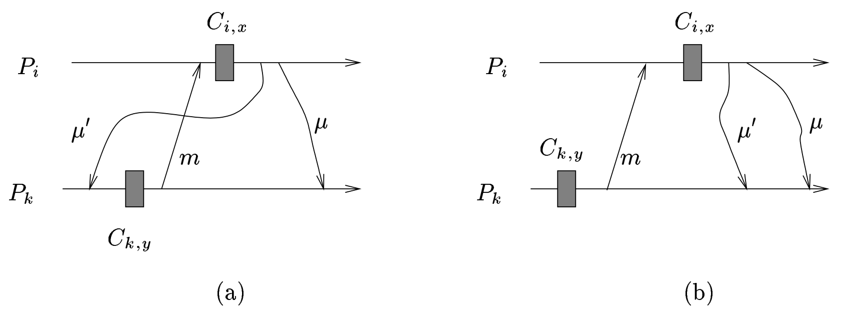


Figure 6: proof of Lemma 4.1.

The previous lemma says that if a checkpoint is involved in a Z-cycle whose chain ζ has size one, then there exists a PZC involving the same checkpoint. The following lemma extends the previous result to a chain ζ of any size:

Lemma 4.2
If *there exists* $ZC(C_{i,x}, \mu, I_{k,y}, \zeta)$
then *there exists* $PZC(C_{i,x}, \mu', I_{l,z}, \zeta')$ *with* $|\zeta'| \leq |\zeta|$.

Proof
Let us consider $ZC(C_{i,x}, \mu, I_{k,y}, \zeta)$. We have two alternatives:

1 **if** $|\zeta| = 1$ **then** the claim follows from Lemma 4.1;

2 **if** $|\zeta| > 1$ **then** if $\mu \in min(M(C_{i,x}, P_k))$ then the claim trivially follows when considering $\mu' = \mu$. Otherwise let us consider $\mu' \in min(M(C_{i,x}, P_k))$ (note that μ' exists as $M(C_{i,x}, P_k)$ is not empty since it contains μ).

There are two cases:

2.1 $send(\zeta.first)\overset{e}{\rightarrow}deliver(\mu'.last)$ (see Figure 7.a).

In this case we get $PZC(C_{i,x}, \mu', I_{k,y}, \zeta)$ and the claim follows;

2.2 $deliver(\mu'.last)\overset{e}{\rightarrow}send(\zeta.first)$ (see Figure 7.b).

In this case, by construction, we get $ZC(C_{i,x}, [\mu' \circ \mu''], I_{h,w}, \zeta')$ where $\zeta = \mu'' \overset{h,w}{\bullet} \zeta'$. Note that $|\mu''| \geq 1$ and $|\zeta'| < |\zeta|$ (see Figure 7.c).

If we fall in case 2.2, the previous construction can be repeated on $ZC(C_{i,x}, [\mu' \circ \mu''], I_{h,w}, \zeta')$ and after a finite number of steps either we fall in case 2.1 or we get $ZC(C_{i,x}, \widehat{\mu}, I_{l,z}, \widehat{\zeta})$ with $|\widehat{\zeta}| = 1$ thus the claim follows from Lemma 4.1.

□

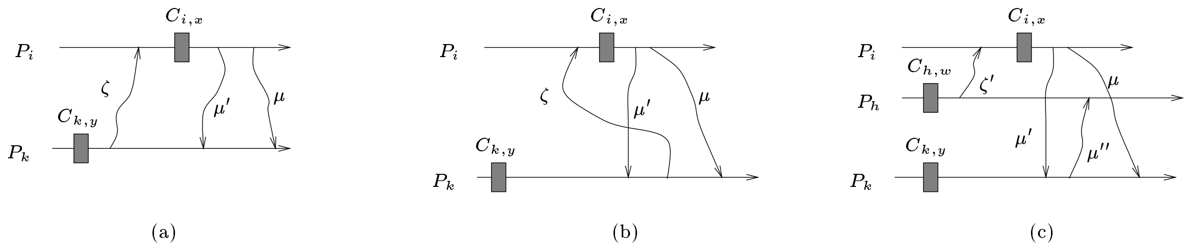


Figure 7: proof of Lemma 4.2.

4.2 Core Z-Cycles

This paragraph introduces the notion of Core Z-Cycle (CZC). It is interesting because of the result in Lemma 4.4 which states that if there is a PZC involving a checkpoint then there exists a CZC that involves a checkpoint (not necessarily the same checkpoint involved in the PZC). A CZC is actually a PZC with a restriction on its structure. This restriction is on the sequence of checkpoint intervals related to its message chain ζ as can be seen from the following definition:

Definition 4.2 *(Core Z-Cycle)*
Let consider $PZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ and let $S(\zeta)$ be the sequence of checkpoint intervals associated with ζ. $PZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ is a Core Z-Cycle, denoted $CZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ if, and only if:

$$\forall I_{j_i,z_i} \in S(\zeta) \Rightarrow \neg(I_{j_i,z_i+1} \overset{I}{\rightarrow} I_{k,y})$$

Figure 8 shows an example of a CZC involving $C_{i,x}$ and an example of a PZC which is not a CZC as it contradicts the restriction in Definition 4.2 (i.e., $I_{j,z+1} \overset{I}{\rightarrow} I_{k,y}$ due to the presence of the causal message chain μ'). Note that, in the latter case, $PZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ embeds the Z-cycle $ZC(C_{j,z+1}, \mu', I_{k,y}, (\zeta - \zeta.last))$ as shown in Figure 8.b. This recursive behavior will be exploited in the proof of Lemma 4.4.

Let us now prove that if there exists a PZC in a checkpoint and communication pattern of a distributed execution, then there exists a CZC in that pattern, assuming the size of the non-causal message chain of the PZC equal to one, we will then generalize the result to a chain of any size:

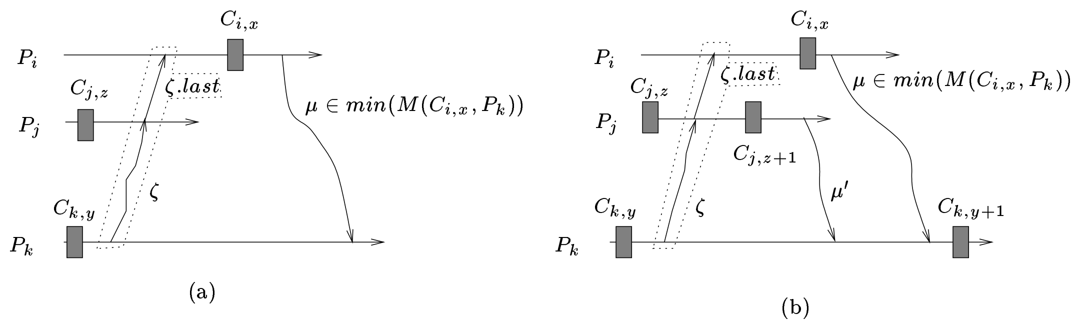


Figure 8: a CZC involving $C_{i,x}$ (a); an example of PZC involving $C_{i,x}$ which is not a CZC (b).

Lemma 4.3

If *there exists* $PZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ *such that* $|\zeta| = 1$

then *there exists* $CZC(C_{i,x}, \mu, I_{k,y}, \zeta)$.

Proof (By Contradiction)

Let us consider $PZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ with $\zeta = m$ (i.e., $|\zeta| = 1$) and suppose that $CZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ does not exist. As $m \circ C_{i,x} \circ \mu \overset{k,y}{\bullet} m$, $send(m) \in I_{k,y}$ and $\mu \in min(M(C_{i,x}, P_k))$, there must exist $C_{k,y+1}$ such that:

$$I_{k,y+1} \xrightarrow{I} I_{k,y}$$

In this case, by Definition 2.3, there must exist an event $e' \in I_{k,y+1}$ and an event $e'' \in I_{k,y}$ such that $e' \xrightarrow{e} e''$ which is not possible due to the fact that the $\xrightarrow{e}$ relation is acyclic.

□

Lemma 4.4

If *there exists* $PZC(C_{i,x}, \mu, I_{k,y}, \zeta)$

then *there exists a CZC.*

Proof

Let us consider $PZC(C_{i,x}, \mu, I_{k,y}, \zeta)$. We have two alternatives:

1 **if** $|\zeta| = 1$ **then** the claim follows from Lemma 4.3;

2 **if** $|\zeta| > 1$ **then** let us consider the sequence of checkpoint intervals $S(\zeta)$. There are two cases:

2.A $\forall I_{j_i,z_i} \in S(\zeta) \ \Rightarrow \ \neg(I_{j_i,z_i+1} \xrightarrow{I} I_{k,y})$.
By definition 4.2, we get $CZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ and the claim follows;

2.B $\exists I_{j_i,z_i} \in S(\zeta) \ : \ I_{j_i,z_i+1} \xrightarrow{I} I_{k,y}$.
Let $I_{j,z}$ be the *first* checkpoint interval in $S(\zeta)$ satisfying the condition of Case 2.B. There exists at least one causal message chain starting after $C_{j,z+1}$ and ending in $I_{k,y}$ or

in a previous checkpoint interval of P_k. Therefore, the set $M(C_{j,z+1}, P_k)$ is not empty. Let us consider $\mu' \in min(M(C_{j,z+1}, P_k))$; we have two cases:

2.B.1 $send(\zeta.first) \overset{e}{\rightarrow} deliver(\mu'.last)$ (see Figure 9.a).
We get $ZC(C_{j,z+1}, \mu', I_{k,y}, \zeta^*)$ where $\zeta^* = \zeta - \widehat{\zeta}$ and $send(\widehat{\zeta}.first) \in I_{j,z}$. From Lemma 4.2, there exists $PZC(C_{j,z+1}, \bar{\mu}, I_{l,t}, \bar{\zeta})$ with $|\bar{\zeta}| \leq |\zeta^*| < |\zeta|$;

2.B.2 $deliver(\mu'.last) \overset{e}{\rightarrow} send(\zeta.first)$ (see Figure 9.b).
We get $ZC(C_{j,z+1}, [\mu' \circ \mu''], I_{b,s}, \zeta')$ where $\mu'' \overset{b,s}{\bullet} \zeta' = \zeta - \widehat{\zeta}$ and $send(\widehat{\zeta}.first) \in I_{j,z}$, hence $|\zeta'| < |\zeta|$ (see Figure 9.c). By Lemma 4.2 there exists $PZC(C_{j,z+1}, \bar{\mu}, I_{l,t}, \bar{\zeta})$ with $|\bar{\zeta}| \leq |\zeta'|$. So we have $|\bar{\zeta}| < |\zeta|$;

In both cases we obtain a PZC with $|\bar{\zeta}| < |\zeta|$.

If we fall in case 2.B, the previous construction can be applied on the obtained PZC. After a finite number of steps, either we fall in case 2.A or $|\bar{\zeta}| = 1$ thus, by Lemma 4.3, we get a CZC.

□

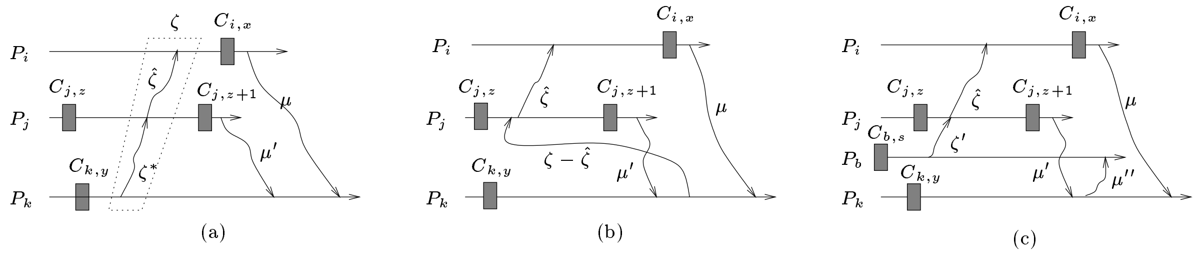


Figure 9: proof of Lemma 4.4.

4.3 A Characterization Theorem

Let us formally introduce the No-Core-Z-Cycle property $\mathcal{NCZC}$:

Definition 4.3
A checkpoint and communication pattern $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$ of a distributed execution satisfies the No-Core-Z-Cycle ($\mathcal{NCZC}$) property if, and only if, no CZC exists in $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$

The following characterization theorem is straightforwardly derived from lemmas introduced in previous sections:

Theorem 4.5
A checkpoint and communication pattern $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$ of a distributed execution satisfies the $\mathcal{NZC}$ property if, and only if, $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$ satisfies the $\mathcal{NCZC}$ property.

Proof

If part. By Lemma 4.2 if a ZC exists then a PZC exists in $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$. By lemma 4.4 if a PZC exists then a CZC exists in $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$. Thus, in terms of properties, $\neg(\mathcal{NZC}) \Rightarrow \neg(\mathcal{NCZC})$. Hence $\mathcal{NCZC} \Rightarrow \mathcal{NZC}$.

Only if part. If the $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$ satisfies $\mathcal{NZC}$ then no CZC exists as CZCs are Z-cycles. So $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$ satisfies $\mathcal{NCZC}$.

□

5 Preventing Core Z-Cycles On-The-Fly

In this section we first introduce a particular family, namely $\mathcal{F}_{\mathcal{NZC}}$, of checkpointing protocols ensuring $\mathcal{NZC}$. Then we show that a protocol in $\mathcal{F}_{\mathcal{NZC}}$ cannot track CZCs on-the-fly. Thus we introduce the notion of *Suspect Core Z-Cycle* (SCZC) which is a checkpoint and communication pattern whose structure corresponds to the *causal part of any CZC.* This pattern can be tracked on-the-fly by protocols in $\mathcal{F}_{\mathcal{NZC}}$. Finally a checkpointing protocol that tracks and prevents all SCZCs is given.

5.1 A Checkpointing Protocol Family $\mathcal{F}_{\mathcal{NZC}}$

We assume the existence of three layers ([1]): application layer, checkpointing protocol layer and communication system layer. Each process is an instance of the checkpointing protocol source code. Messages arrive at processes from a communication system and they will be delivered to the application layer. The application layer generates events of sending messages and of taking checkpoints (basic checkpoints) to processes. Send events are delivered to the communication system. A process takes either basic checkpoints or forced checkpoints. The latter ones are triggered by a *predicate* evaluated at the time a message is received from the communication system layer.

A checkpointing protocol belongs to $\mathcal{F}_{\mathcal{NZC}}$ if (i) it ensures that the final checkpoint and communication pattern of a distributed execution $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$ produced by that protocol satisfies the $\mathcal{NZC}$ property and (ii) the protocol respects the following constraints:

C1. The usable knowledge at an event e is the restriction of $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$ to e's *causal past* (the causal past of an event e is the set $\mathcal{H}_e = \{e' \in \mathcal{H} | e' \xrightarrow{e} e\}$);

C2. Upon the arrival of a message m at process P_k, the protocol has to evaluate the predicate *on-the-fly* (i.e., without additional delays). If it is evaluated to true, a forced checkpoint has to be taken before delivering m to the above layer;

C3. The evaluation of the predicate is based on the usable knowledge available at that event (i.e., the local context of the process plus the control information piggybacked on the application message). In other words, no "control" message is allowed;

C4. All the communication events arrived from the communication system (resp. application layer), are processed by the protocol and delivered to the application layer (resp. communication system) in the same order they arrived. Every event of taking a basic checkpoint is executed by the protocol;

C5. The content of an application message cannot be interpreted by the checkpointing protocol;

C6. Information about other processes (such as clock speed, clock drift, etc.) and about the network's characteristics (such as the maximum message transmission delay) are not known by any process.

As an example the checkpointing protocols FDAS and FDI, presented in [23] and the ones shown in [2], [4], [11], and [20] belong to $\mathcal{F}_{\mathcal{NZC}}$ whereas coordinated checkpointing protocols [5, 13], the ones where the action to take a forced checkpoint is triggered by a send event (as an example the CAS and the CASBR protocols described in [23]), the ones that reorder message deliveries [24], the ones that invalidate the action to take a basic checkpoint [3, 16], the ones that assume a maximum drift rate between any pair of physical clocks accessed by processes [6] and the ones that exploits the semantic of a message content [15] do not belong to $\mathcal{F}_{\mathcal{NZC}}$.

5.2 Suspect Core Z-Cycles

Theorem 4.5 states that a checkpoint and communication pattern of a distributed execution $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$ satisfies the $\mathcal{NZC}$ property if, and only if, no CZC exists in $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$. $CZC(C_{i,x}, \mu, I_{k,y}, \zeta)$ can be broken by placing an additional local checkpoint of process P_k taken between the send of $\zeta.first$ and the delivery of $\mu.last$ as shown in Figure 10.a. The instant of time before the event $deliver(\mu.last)$ represents "the last opportunity" for taking an additional (forced) checkpoint in order to remove that CZC. A CZC is trackable on-the-fly at the last opportunity by a protocol in $\mathcal{F}_{\mathcal{NZC}}$ only if its chain ζ is causal. If ζ contains one or more non-causal concatenations, the CZC becomes no longer trackable.

The previous argument shows that the best a protocol in $\mathcal{F}_{\mathcal{NZC}}$ can do to *prevent* the formation of core Z-cycles is to remove those checkpoint and communication patterns whose structure represents *the common causal part of any core Z-cycle* which is detectable by a process at the last opportunity. Those considerations lead to the introduction of a checkpoint and communication pattern, namely Suspect Core Z-Cycle (SCZC), which is trackable by a protocol in $\mathcal{F}_{\mathcal{NZC}}$:

Definition 5.1
A Suspect Core Z-Cycle (SCZC) is a checkpoint and communication pattern $SCZC(I_{j,z}, C_{i,x}, \mu, I_{k,y})$ such that:

$$\exists m, m' \ : \ C_{j,z} \circ m \circ C_{i,x} \circ \mu \overset{k,y}{\bullet} m' \ \ \mathit{with} \ \begin{cases} (i) & send(m) \in I_{j,z} \\ (ii) & \mu \in min(M(C_{i,x}, P_k)) \\ (iii) & \nexists e \in I_{j,z+1} \ \ : \ \ e \overset{e}{\rightarrow} deliver(\mu.last) \end{cases}$$

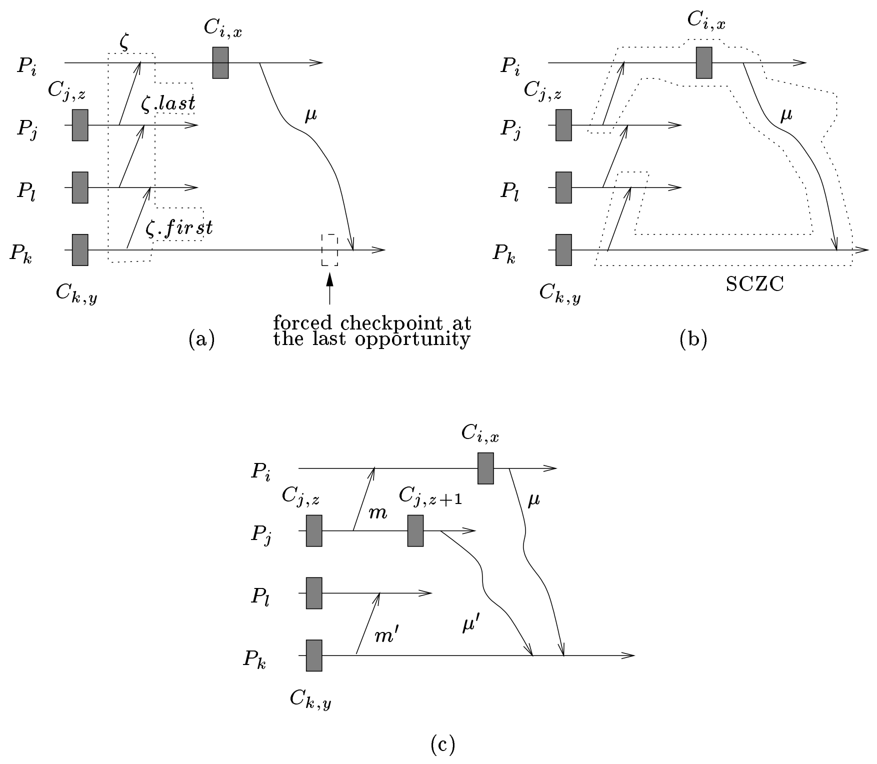


Figure 10: an example of CZC non-trackable on-the fly by a protocol in $\mathcal{F}_{NZC}$ (a); an example of SCZC pattern (b); a pattern which is not an SCZC (c).

As an example $SCZC(I_{j,z}, C_{i,x}, \mu, I_{k,y})$ is shown in Figure 10.b while Figure 10.c shows a checkpoint and communication pattern which is not an SCZC as it violates the constraint (iii) of previous definition (due to the causal message chain μ').

Let us now state a theorem, actually a sufficient condition for the $\mathcal{NZC}$ property, that will be used to design the protocol of the next section:

Theorem 5.1

If a checkpoint and communication pattern of a distributed execution $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$ does not include any SCZC (i.e., it satisfies the No-Suspect-Core-Z-Cycle property $\mathcal{NSCZC}$) then $(\widehat{\mathcal{H}}, \mathcal{C}_{\widehat{\mathcal{H}}})$ satisfies the $\mathcal{NZC}$ property.

Proof

From the structure of the CZC and of the SCZC, it trivially follows, in terms of properties, $\mathcal{NSCZC} \Rightarrow \mathcal{NCZC}$. From Theorem 4.5 we have $\mathcal{NCZC} \Rightarrow \mathcal{NZC}$. Hence we get $\mathcal{NSCZC} \Rightarrow \mathcal{NZC}$.
$\square$

The reader could now wonder if the SCZC is the right pattern to prevent in order to remove CZCs. In particular, why does the SCZC structure include only the last checkpoint interval passed through by ζ (i.e., $I_{j,z}$) and not all the checkpoint intervals associated with the *final causal part* of ζ. This causal part would represent the largest part of ζ visible by P_k at the last opportunity.

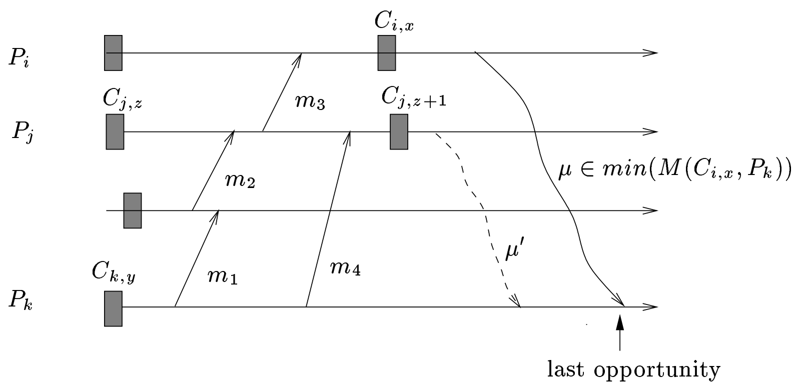


Figure 11: a set of PZCs involving $C_{i,x}$.

Let **MC** be the set of message chains ζ starting after $C_{k,y}$, terminating before $C_{i,x}$ and sharing the last message $\zeta.last$. This defines a set **X** of PZCs, one for each distinct ζ in **MC**. If we consider Z-cycles involving $C_{i,x}$ in Figure 11 we have **MC**$= \{[m_1, m_2, m_3], [m_4, m_3]\}$ and **X**$= \{PZC(C_{i,x}, \mu, I_{k,y}, [m_1, m_2, m_3]), PZC(C_{i,x}, \mu, I_{k,y}, [m_4, m_3])\}$.

Let us assume the existence of the causal message chain μ' (depicted by a dotted line). As a consequence we have $\exists e \in I_{j,z+1} : \ e \to deliver(\mu.last)$ which implies $I_{j,z+1} \xrightarrow{I} I_{k,y}$. Hence, each PZC in **X** is not a CZC (see Definition 4.2).

If μ' does not exist, P_k cannot safely conclude at the last opportunity that any CZC involving $C_{i,x}$ can never be formed. As an example, a message m_4 sent in the interval $I_{k,y}$ could be received by P_j in $I_{j,z}$ after the sending of m_3. This non-causal concatenation is out of the usable knowledge of P_k and gives rise to $CZC(C_{i,x}, \mu, I_{k,y}, [m_4, m_3])$. Hence, a protocol in $\mathcal{F}_{\mathcal{NZC}}$ directs a forced checkpoint before executing $deliver(\mu.last)$ if no information concerning the closure of the checkpoint interval $I_{j,z}$ has been sent to P_k by means of a causal message chain.

5.2.1 A Remark on Characterizations Stronger than $\mathcal{NCZC}$

Imposing additional constraints on the structure of a CZC can lead to characterizations stronger than $\mathcal{NCZC}$. As an example, let us consider the subset **X** of **CZC** such that (i) the length of μ is minimal, (ii) ζ is a member of a set of message chains that establish the first Z-path between $C_{k,y}$ and $C_{i,x}$ (this set contains message chains sharing the last message), and (iii) ζ is the chain with minimal length in that set[5]. The existence of any CZC implies the existence of a Z-cycle in **X**, thus, if **X** is empty, then **CZC** is empty.

Although the latter characterization could be interesting from a theoretical point of view, from a practical one, it does not add information, suitable for protocols in $\mathcal{F}_{\mathcal{NZC}}$, in order to reduce the number of forced checkpoints compared to the one provided by CZC. In other words, this characterization does not help to find checkpoint and communication patterns more refined than SCZC and detectable on-the-fly. More specifically, the information concerning the "time" at which

[5]In such a case we have a "temporal" and a "spatial" constraint both on ζ and on μ.

the chain ζ is established and the length of ζ does not help as ζ is, usually, non-causal and, thus, it cannot be tracked at the last opportunity as shown in the previous section. The information on the length of μ does not help to save forced checkpoints as the concept of *min* is related to a set of causal message chains (see Section 4.1) which includes the one of minimal length. Therefore, preventing a non-causal concatenation (e.g. $\mu \bullet m$) due to either any chain of the set $min(M(C_{i,x}, P_k))$ or to the one with minimal length has the same effect in terms of forced checkpoints.

5.3 A Checkpointing Protocol in $\mathcal{F}_{\mathcal{NZC}}$

The protocol presented in this section tracks on-the-fly all the SCZC patterns, and breaks them by introducing forced checkpoints before delivering message $\mu.last$ (i.e., at the last opportunity). This is done by exploiting the control information piggybacked on application messages, that encodes the causal past with respect to the event of the delivery of a message, and the local history of a process (i.e., fully exploiting the usable knowledge at that event). The protocol uses a vector clock and a matrix of integers as control information.

5.3.1 Tracking SCZC Patterns

In order to track the formation of $SCZC(I_{j,z}, C_{i,x}, \mu, I_{k,y})$, upon the arrival of a message $\mu.last$, process P_k has to verify whether conditions for the existence of that checkpoint and communication pattern are satisfied. In the following paragraphs we introduce the data structures to accomplish this task.

Tracking $\mu \in min(M(C_{i,x}, P_k))$.
To detect if $\mu \in min(M(C_{i,x}, P_k))$, a vector clock mechanism is used considering checkpoints of processes as relevant events [17]. Each process P_k maintains a vector clock VC_k whose size corresponds to the number of processes. $VC_k[i]$ stores the maximum checkpoint rank of P_i seen by P_k and $VC_k[k]$ stores the rank of the last checkpoint taken by P_k. VC_k is initialized to zero except the k-th entry which is initialized to one. Each application message `m` sent by P_k piggybacks the current value of VC_k (denoted $\mathtt{m}.VC$). Following the updating rule of a vector clock, upon the delivery of a message `m`, VC_k is updated from $\mathtt{m}.VC$ by taking a component-wise maximum.

A causal message chain μ including message `m` as $\mu.last$ is prime (i.e., μ belongs to some $min(M(C_{i,*}, P_k))$, if, upon the delivery of `m` to process P_k, the following predicate holds ([6]):

$$\exists i \ : \ (\mathtt{m}.VC[i] > VC_k[i])$$

Tracking $\mu \overset{k,y}{\bullet} m'$.
To detect if there exists a non-causal concatenation between a prime causal message chain μ and a message m' in the interval $I_{k,y}$, process P_k maintains a boolean variable $after\_first\_send_k$. This

[6]We use the term "prime" for a chain $\mu \in min(M(C_{i,*}, P_k)$ as that chain can be the first one bringing to P_k the knowledge of the existence of new checkpoints of P_i.

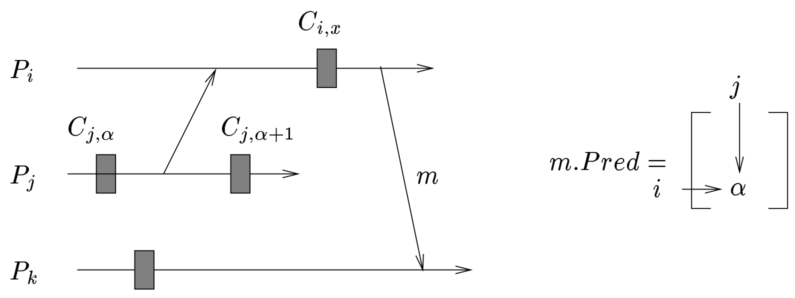


Figure 12: an example of values stored in $m.Pred$.

variable is set to TRUE when a send event occurs. It is set to FALSE each time a local checkpoint is taken. Hence, upon the delivery of a message m (with m $= \mu.last$), P_k detects that $\mu \overset{k,y}{\bullet} m'$ if the following predicate hold:

$$after\_first\_send_k \wedge (\exists i \ : \ (\mathtt{m}.VC[i] > VC_k[i]))$$

Tracking $C_{j,z} \circ m \circ C_{i,x}$**.**
Each process P_k maintains a vector of integers Imm_Pred_k of size n and a matrix of integers $Pred_k$ of size $n \times n$. $Imm\_Pred_k[\ell]$ represents the maximum rank of the checkpoint interval from which process P_ℓ sent a message m which has been delivered by P_k in its current checkpoint interval $I_{k,y-1}$ (in other words $C_{\ell,Imm\_Pred[\ell]}$ is an immediate predecessor of checkpoint $C_{k,y}$). Each entry of this vector is set to -1 every time a checkpoint is taken by P_k.

$Pred_k[i,j]$ represents, to the knowledge of P_k, the maximum rank of the checkpoint interval from which process P_j sent a message m which has been delivered by P_i in a checkpoint interval $I_{i,x-1}$ with $x \leq VC_k[i]$. The matrix $Pred_k$ is initialized to -1, its content is piggybacked on each message m sent by P_k (m.$Pred$) and the rules to update its entries are the following:

1. Whenever a checkpoint is taken by P_k, $Pred_k[k,-]$ is updated as follows:

$$\forall j \ \ Pred_k[k,j] = max(Pred_k[k,j], Imm\_Pred_k[j])$$

2. Upon the arrival of a message m at P_k:

$$\forall \ell, t \ \ Pred_k[\ell,t] = max(Pred_k[\ell,t], \mathtt{m}.Pred[\ell,t])$$

Figure 12 shows an example of a checkpoint and communication pattern and the content of $m.Pred[i,j]$ associated with that pattern.

Tracking $\nexists e \in I_{j,z+1} \ : \ e \xrightarrow{e} deliver(\mu.last)$**.**
Upon the arrival of a message m ending a prime causal chain (i.e., $\exists i \ : \ (\mathtt{m}.VC[i] > VC_k[i])$), in order to track the above condition, we need to know if there exists a j such that m.$Pred[i,j] + 1$

does not belong to the causal past of the delivery of m. This knowledge is encoded in $\mathtt{m}.VC[j]$ and $VC_k[j]$. Hence, the predicate becomes:

$$\exists j \ : \ \mathtt{m}.Pred[i,j]+1 > max(\mathtt{m}.VC[j], VC_k[j])$$

```
init P_k:
take a checkpoint;
$after\_first\_send_k := FALSE$;
$\forall i \ : \ i \neq k \ VC_k[i] := 0; \ VC_k[k] := 1$;
$\forall i, \forall j \ Pred_k[i,j] := -1; \ \forall h \ Imm\_Pred_k[h] := -1$;

when m arrives at $P_k$ from $P_l$:
if $after\_first\_send_k \wedge (\exists i \ : \ (\mathtt{m}.VC[i] > VC_k[i]) \wedge$
    $(\exists j \ : \ \mathtt{m}.Pred[i,j]+1 > max(\mathtt{m}.VC[j], VC_k[j])))$
then $take\_ckpt()$;    % forced checkpoint %
$\forall i \ VC_k[i] := max(VC_k[i], \mathtt{m}.VC[i])$;
$\forall i, \forall j \ Pred_k[i,j] := max(\mathtt{m}.Pred[i,j], Pred_k[i,j])$;
$Imm\_Pred_k[l] := max(Imm\_Pred_k[l], \mathtt{m}.VC[l])$;

procedure $send(\mathtt{m}, P_j)$:
$\mathtt{m}.content = data; \ \mathtt{m}.VC := VC_k; \ \mathtt{m}.Pred := Pred_k$;
send m to $P_j$;
$after\_first\_send_k := TRUE$;

when a basic checkpoint is scheduled from $P_k$:
$take\_ckpt()$;

procedure $take\_ckpt()$:
take a checkpoint;
$\forall h \ Pred_k[k,h] := max(Pred_k[k,h], Imm\_Pred_k[h])$;
$\forall h \ Imm\_Pred_k[h] := -1$;
$VC_k[k] := VC_k[k]+1$;
$after\_first\_send_k := FALSE$;
```

Figure 13: the protocol.

5.3.2 Preventing SCZC Patterns

Upon the arrival of a message m at process P_k in $I_{k,y}$, if the following predicate holds:

$$after\_first\_send_k \wedge (\exists i \ : \ (\mathtt{m}.VC[i] > VC_k[i]) \wedge (\exists j \ : \ \mathtt{m}.Pred[i,j]+1 > max(\mathtt{m}.VC[j], VC_k[j])))$$

then, process P_k detects that at least one $SCZC(I_{j,Pred_k[i,j]}, C_{i,x}, \mu, I_{k,y})$ is going to be formed with $\mathtt{m} = \mu.last$ and $VC_k[i] < x \leq \mathtt{m}.VC[i]$. In this case P_k directs a forced checkpoint $C_{k,y+1}$ before the delivery of m. The behavior of process P_k is shown in Figure 13 (all the procedures and the message handler are executed in atomic fashion).

We would like finally to remark that, from an operational point of view, the elements of the diagonal of the matrix $Pred$ are never used by the protocol. Hence, when implementing the protocol, the vector clock VC can be embedded in that diagonal. Thus, the resulting control information piggybacked on application messages boils down to a matrix of $n \times n$ integers.

6 A Taxonomy of Checkpointing Protocols in $\mathcal{F}_{\mathcal{NZC}}$

In this section we describe the *Virtual Precedence* ($\mathcal{VP}$) property introduced by Helary et al. in [11]. Then we present a taxonomy of checkpointing protocols in $\mathcal{F}_{\mathcal{NZC}}$. This taxonomy splits protocols in two classes: $\mathcal{VP}$*-enforced* and $\mathcal{VP}$*-accordant.*

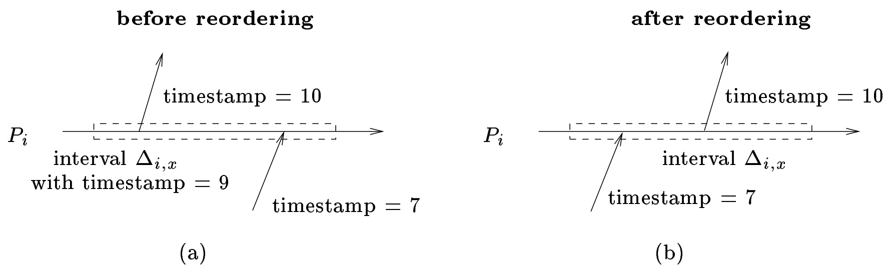


Figure 14: the Virtual Precedence property.

6.1 The Virtual Precedence Property

In a seminal paper ([11]) Helary et al. have shown that ensuring $\mathcal{NZC}$ in a checkpoint and communication pattern of a distributed execution is a particular application of a property, namely Virtual Precedence ($\mathcal{VP}$), defined on an interval-based abstraction of the distributed execution.

Informally, denoting with $\Delta_{i,x}$ the x-th interval of process P_i, the abstraction satisfies $\mathcal{VP}$ if, and only if, it is possible to timestamp messages and intervals in a way that:

- for any pair of messages m and m' such that $deliver(m) \in \Delta_{i,x}$ and $send(m') \in \Delta_{i,x}$ then the timestamp of m is smaller than or equal to the timestamp of m';
- the timestamp of $\Delta_{i,x}$ is larger than or equal to the timestamp of all messages delivered in $\Delta_{i,x}$ and is smaller than or equal to the timestamp of all messages sent in $\Delta_{i,x}$.

This means that, in the logical time (timestamp), communications can be seen as causal in each interval. That is, communication events can be reordered in any interval making all deliver events to precede all send events and timestamp does not decrease along any causal message chain. An example of this is shown in Figure 14.

In other words, an interval-based abstraction of a distributed execution satisfies $\mathcal{VP}$ if, and only if, it is possible to associate a *timestamping function* with intervals having the following features: (F1) intervals which are connected by a message must be timestamped in a non-decreasing way (safety part) and (F2) the timestamp of a process must increase after communication (liveness part). It is easy to check that if we consider each single event as an interval of the abstraction, the timestamping function boils down to the Lamport's scalar clock [14] or the Fidge-Mattern's vector time [8, 17]. In the particular context of the checkpointing problem, intervals correspond to checkpoint intervals (i.e., $\Delta_{i,x} \equiv I_{i,x}$) and, then, the abstraction of the distributed execution corresponds to a checkpoint and communication pattern. Helary et al. [11] proved a theorem that, in the context of checkpointing, can be stated as follows:

Theorem 6.1 *(Helary et al. [11])*
A checkpoint and communication pattern of a distributed execution ($\widehat{\mathcal{H}}$,$\mathcal{C}_{\widehat{\mathcal{H}}}$) satisfies $\mathcal{VP}$ if, and only if, it satisfies $\mathcal{NZC}$ (i.e., $\mathcal{VP} \Leftrightarrow \mathcal{NZC}$).

In other words, $\mathcal{VP}$ constitutes a common basis for all checkpointing protocols ensuring $\mathcal{NZC}$. We show, however, that protocols in $\mathcal{F}_{\mathcal{NZC}}$ can be split into two sub-families according to the way they "see" the previous equivalence between $\mathcal{VP}$ and $\mathcal{NZC}$.

6.2 $\mathcal{VP}$-Enforced Checkpointing Protocols

Let us assume the existence of a function timestamping checkpoint intervals consistently with F1 and F2. A checkpointing protocol in $\mathcal{F}_{\mathcal{NZC}}$ can be then derived as follows. Timestamps are piggybacked on any application message. Upon the arrival of a message m at P_i in $I_{i,x}$, a forced checkpoint will be taken if the delivery of the message would violate F1 or F2. The checkpoint interval $I_{i,x+1}$ is then timestamped by the protocol according to the timestamping function.

We call $\mathcal{VP}$-enforced protocol any protocol relying on an *a priori* timestamping function. Examples of such protocols are in [4, 11, 12, 22]. In what follows we discuss the Briatico et al. protocol [4] (BCS) and a protocol based on vector times.

The BCS Protocol [4]. In this protocol, an integer is assumed to timestamp checkpoint intervals. Thus, each process P_k endows a variable (the timestamp) denoted ts_k. The timestamp is managed by P_k according to the following rules:

1. when starting the execution, ts_k is initialized to zero;
2. when sending a message m, a copy of ts_k is piggybacked on m (denoted m.ts);
3. when taking a basic checkpoint, $ts_k := ts_k + 1$;
4. when a message m arrives at P_k, if m.$ts > ts_k$ then a forced checkpoint is taken by P_k and $ts_k :=$ m.ts.

Other protocols have been presented after the BCS protocol; they lay on more refined timestamp management rules that allow the number of forced checkpoints to be reduced [11, 12].

A Vector Time Based Protocol. Assume using a vector time [8, 17] to timestamp checkpoint intervals when considering those intervals as vector time relevant events. Each process P_k endows a vector of integers TS_k (the timestamp) where $TS_k[i]$ represents the highest checkpoint interval of process P_i *seen* (directly or transitively) by process P_k. The vector is updated by process P_k according to the following rules:

1. when starting the execution, TS_k is initialized to zero, but the k-th entry which is set to one;
2. when sending a message m, a copy of TS_k is piggybacked on m (denoted m.TS);
3. when taking a basic checkpoint, $TS_k[k] := TS_k[k] + 1$;
4. when a message m arrives at P_k, if $\exists j$: m.$TS[j] > TS_k[j]$ then a forced checkpoint is taken by P_k and $\forall i\ TS_k[i] := max(TS_k[i],$m.$TS[i])$.

This protocol corresponds exactly to the one proposed by Vankatesh et al. in [22]. It is interesting to remark that if we consider each process takes a basic checkpoint after each event, the timestamp of the BCS protocol and the Venkatesh et al. vector boil down to the Lamport's scalar clock and the Fidge-Mattern's vector time respectively.

Finally, let us note that, in some sense, $\mathcal{VP}$-enforced corresponds to that protocols' class that sees the relation of equivalence of Theorem 6.1 from left to right. i.e., first ensuring $\mathcal{VP}$. As a consequence $\mathcal{NZC}$.

6.3 $\mathcal{VP}$-Accordant Checkpointing Protocols

A $\mathcal{VP}$-accordant protocol prevents the formation of a specific checkpoint and communication pattern which, in turn, avoids the occurrence of Z-cycles, i.e., the predicate that triggers the action to take a forced checkpoint depends on the checkpoint and communication patterns that are going to be formed if a message would be delivered. Thus, if the predicate is evaluated to true, at least one "bad" checkpoint and communication pattern is going to be formed. Then the protocol takes a forced checkpoint to break that pattern.

As $\mathcal{VP} \Leftrightarrow \mathcal{NZC}$, also for a $\mathcal{VP}$-accordant protocol there will exist a timestamping function that could be used to timestamp checkpoint intervals produced by the protocol consistently with F1 and F2. However such a function does not play any role in the design of the protocol. Examples of $\mathcal{VP}$-accordant protocols are in [2, 20, 23]. Some of them are discussed below.

The RUS Protocol [20]. This protocol accepts only causal message chains. It actually prevents the formation of $\langle send \cdot deliver \rangle$ (i.e., $m \bullet m'$) patterns in any checkpoint interval by means of forced checkpoints, so no non-causal concatenation of messages can ever occur, preventing the formation of Z-cycles.

The FDAS Protocol [23]. FDAS avoids the formation of checkpoint and communication patterns with the following structure $C_{i,x} \circ \mu \overset{k,y}{\bullet} m'$ with $\mu \in min(M(C_{i,x}, P_k))$. As the previous pattern is part of the structure of a PZC, the prevention of all those patterns guarantees the absence of prime Z-cycles and then the $\mathcal{NZC}$ property. FDAS attaches a vector of checkpoint ranks to each application message to check if that bad pattern is going to be formed. The vector can be used to timestamp checkpoint intervals consistently with F1 and F2.

The BHMR Protocol [2]. This protocol prevents the formation of dependencies between two checkpoints due to non-causal message chains composed by two causal message chains (i.e., $\zeta = \mu \overset{k,y}{\bullet} \mu'$) if they are not doubled, in a visible way, by a causal message chain. In terms of concatenation relations, we get that a dependency due to a non-causal message chain $\zeta = \mu \overset{k,y}{\bullet} \mu'$ is doubled by a causal message chain μ'' if the pair of checkpoints related by ζ is also related by μ'' (i.e., if $C_{i,x} \circ \zeta \circ C_{j,y}$ then $C_{i,x} \circ \mu'' \circ C_{j,y}$). The doubling is *visible* by P_k (the only process able to break ζ) if there exists a causal message chain μ''' such that $\mu'' \circ \mu'''$ is prime (i.e., $\mu'' \circ \mu''' \in min(M(C_{i,x}, P_k))$). It is easy to show that this protocol prevents the formation of any $CZC(C_{i,x}, \mu, I_{k,y}, \zeta)$. In particular there are two cases:

- $\zeta = \mu'$ i.e., ζ is a causal message chain. $CZC(C_{i,x}, \mu, I_{k,y}, \mu')$ is a particular dependency between $C_{i,x}$ and itself that cannot be doubled, so it is prevented by taking a forced checkpoint before delivering $\mu.last$;
- $\zeta = \mu_1 \bullet \mu_2 \bullet \ldots \bullet \mu_\ell$ with $\ell > 1$ where each pair of successive causal message chains establishes a dependency between two distinct checkpoints that is non-doubled. Note that, the composition of ζ must exist, otherwise we fall in the previous case. Then the protocol prevents this pattern by taking ℓ forced checkpoints. $\ell - 1$ forced checkpoints are taken to prevent each non-causal concatenation of two successive causal message chains composing ζ. The last forced checkpoint is taken by P_k to prevent the pattern $\mu \overset{k,y}{\bullet} \zeta.first$.

The BHMR protocol ensures $\mathcal{NZC}$ and then $\mathcal{VP}$. It piggybacks a vector of checkpoint intervals and a matrix of booleans on each application message. The vector can be used to timestamp checkpoint intervals consistently with F1 and F2.

The protocol proposed in this paper is a $\mathcal{VP}$-accordant one as it is based on the prevention of suspect core Z-cycle patterns.

6.3.1 A Remark on the Rollback-Dependency-Trackability Property

Differently from our protocol, $\mathcal{VP}$-accordant checkpointing protocols discussed above (namely RUS, FDAS and BHMR) have been designed to ensure the *Rollback-Dependency-Trackability* property ($\mathcal{RDT}$) [23]. This property stipulates that if there exists a Z-path between two checkpoints due to a non-causal message chain ζ (i.e., $C_{i,x} \circ \zeta \circ C_{j,y}$), then there must exist at least one causal message chain μ which establishes a Z-path between those checkpoints (i.e., $C_{i,x} \circ \mu \circ C_{j,y}$).

As the Z-cycle is a particular type of non-causal Z-path between a checkpoint and itself, each checkpoint and communication pattern that satisfies $\mathcal{RDT}$ also guarantees $\mathcal{NZC}$. The vice-versa is not true. Intuitively, this implies that each practical problem solved by $\mathcal{NZC}$ can be solved also by $\mathcal{RDT}$ at the cost of additional overhead.

$\mathcal{RDT}$ allows the design of simple and decentralized solutions for practical problems that need to compute the *minimum or the maximum global consistent checkpoint that includes a given set of checkpoints*, such as software error recovery and output commit. On the other hand, ensuring $\mathcal{NZC}$ is enough to compute *a consistent global checkpoint that includes a given checkpoint*, which suffices for solving important practical problems like domino-free rollback-recovery.

6.4 A Comparison among Checkpointing Protocols

An ideal index for a formal comparison among checkpointing protocols would be the "total number of forced checkpoints" directed by a protocol in a given execution. However, it has been recently proved by Tsai et al. in [21] that there cannot exist an "optimal" protocol with respect to this index. An intuitive explanation of this fact is that the taking of a forced checkpoint influences the future checkpoint and communication patterns of the execution and this influence is unpredictable. To overcome this problem a different criterion has been introduced by Baldoni et al. in [1]. This criterion, namely *the condition criterion*, is based on the usable knowledge (see Section 5.1) of a process at the time the decision to direct a forced checkpoint is taken. More precisely, let us consider two protocols A and B taking a decision to direct a forced checkpoint based on the *same* usable knowledge. We say that "A *is better than* B" if whenever A takes a forced checkpoint, then B takes a forced checkpoint. It has been shown that the condition criterion is strongly correlated with the total number of forced checkpoints [2].

In the rest of the section we compare the performance of our protocol (hereafter P) with RUS, FDAS and BHMR, by using a simulation study and, when possible, the condition criterion. Finally, a simulation study comparing P and BCS, as a member of the $\mathcal{VP}$-enforced family, is presented.

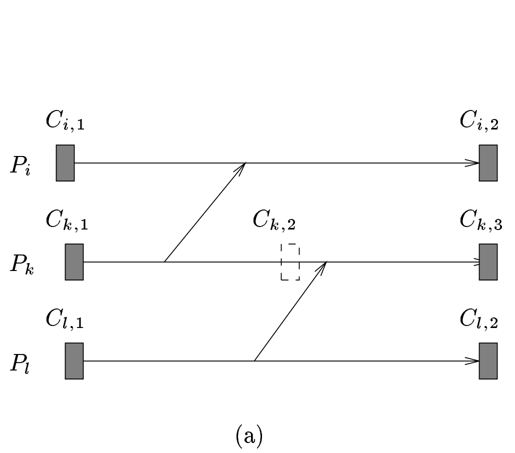
$\mathcal{VP}$-Accordant Checkpointing Protocols. In the context of protocols ensuring $\mathcal{RDT}$ (i.e., RUS, FDAS and BHMR), [1] proves that BHMR *is better than* RUS and FDAS with respect to the condition criterion. As far as P is concerned, the pattern prevented by RUS (i.e., $m \bullet m'$) is a part of an SCZC when considering m as prime. The pattern prevented by FDAS (i.e., $C_{i,x} \circ \mu \overset{k,y}{\bullet} m'$ with $\mu \in min(M(C_{i,x}, P_k))$) is a part of an SCZC. Therefore, also P *is better than* RUS and FDAS with respect to the condition criterion.

To complete the comparison among $\mathcal{VP}$-accordant protocols we have to compare BHMR with P. Unfortunately, they cannot be evaluated with respect to the condition criterion. This is because once the same usable knowledge is fixed, if P takes a forced checkpoint then there is no guarantee that BHMR takes it and vice versa. As an example, Figure 15.a shows a checkpoint and communication pattern in which BHMR takes a forced checkpoint while P does not take it. Figure 15.b shows the opposite scenario. As a consequence, we compared these protocols by means of a simulation study which measured the number of forced checkpoints per message delivery (R) as a function of the average checkpoint interval size (ACI). Experiments were conducted varying ACI from 100 to 10000 events. Two distinct strategies for taking basic checkpoints were considered:

S1 : each process takes N basic checkpoints and the period between two successive basic checkpoints is the same at all processes;

S2 : each process takes N basic checkpoints randomly distributed in the whole execution, with a distinct distribution at each process.

We simulated a point-to-point environment with 8 processes in which each process can send a message to any other and the destination of each message is an uniformly distributed random



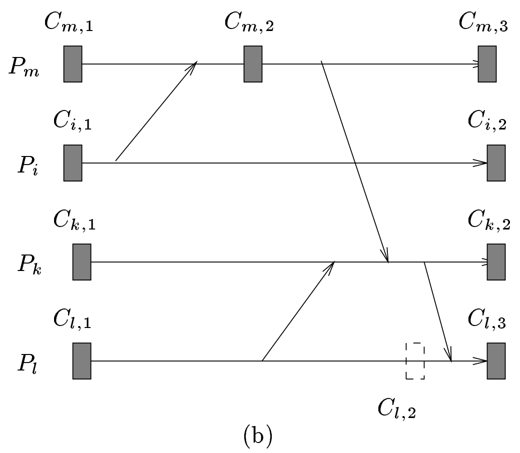
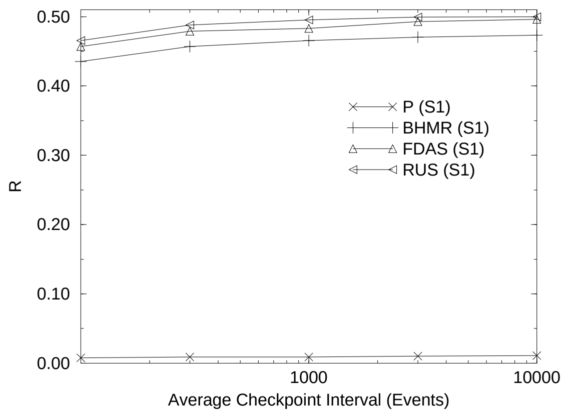


Figure 15: Checkpoint and communication patterns comparing BHMR and P.

variable. Each process executes internal, send and receive operations with probability $p_i = 0.9$, $p_s = 0.05$ and $p_r = 0.05$, respectively. The operation time and the message propagation time are exponentially distributed with mean value equal to 1 and 5 time units respectively. Each simulation run consists of one million events. For each value of ACI we did several simulation runs with different seeds and the results were within five percent of each other, thus, variance is not reported in the plots. As we are interested only in counting how many local states are saved as forced checkpoints per message delivery, the taking of a checkpoint has been simulated as an instantaneous action.



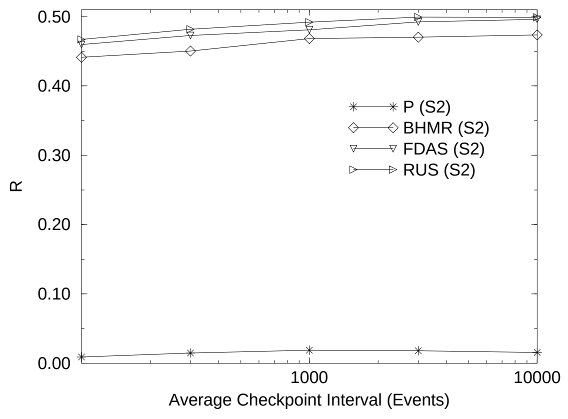


Figure 16: R vs ACI.

Plots in Figure 16 show a gap of an order of magnitude between values of R obtained with protocol P and those obtained with BHMR independently of the used strategy. This deep gap can be easily justified by looking at the checkpoint and communication patterns of Figure 15. The probability of occurrence of patterns like the one of Figure 15.a is much higher compared to the one of Figure 15.b. For the sake of completeness, Figure 16 contains also the curves related to the protocols FDAS and RUS in order to point out the real performance distance compared to P and BHMR.

$\mathcal{VP}$-Enforced Checkpointing Protocols. P and any $\mathcal{VP}$-enforced protocol cannot be evaluated according to the condition criterion as they follow two distinct approaches to avoid Z-cycles. Then, it is easy to find examples of checkpoint and communication patterns in which P takes a forced checkpoint while the $\mathcal{VP}$-enforced protocol does not take it and vice-versa. So, in this case, we also did a comparison between P and BCS (as a member of the $\mathcal{VP}$-enforced family) by means of a simulation study whose results are shown in Figure 17. BCS shows very good performance with a flat behavior of R when adopting S1 as the strategy for taking basic checkpoints. In this case, there is an implicit coordination among processes that ensures the $\mathcal{NZC}$ property by taking only a few forced checkpoints (timestamps increase at the same speed). In the best case no forced checkpoint is taken at all. The value of R for P is flat around 0.01.

Strategy S2 represents a bad scenario for BCS as the distributions of the basic checkpoint events at distinct processes are not correlated. So timestamps increase at different speeds at distinct processes and, then, the performance of BCS depends on ACI as shown in Figure 17 ([7]). P's plot is, also in this case, flat and close to that of strategy S1.

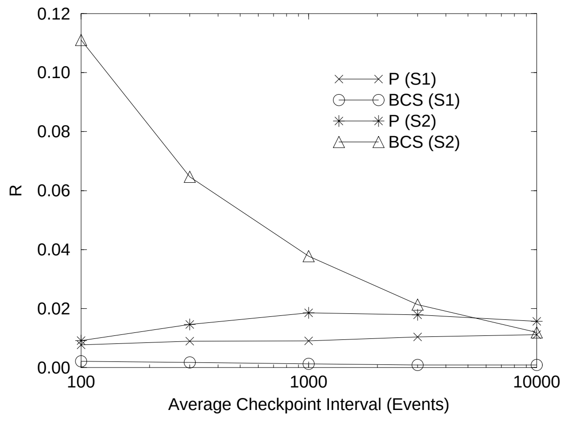


Figure 17: R vs ACI.

From previous plots it can be argued that the performance of P (and in general of any $\mathcal{VP}$-accordant protocol) is more stable compared to the one of BCS with respect to both ACI and the basic checkpointing strategy used. This would suggest employing P in a checkpointing layer of a general purpose system.

[7]In the worst case, if no correlation exists among the strategies for taking basic checkpoints at distinct processes (e.g., random checkpointing strategy), each basic checkpoint directs up to $n-1$ forced checkpoints in other processes.

6.5 A Remark on a Previous Taxonomy

A common intuition separated in the past checkpointing protocols in $\mathcal{F}_{\mathcal{NZC}}$ into two families: *model-based* and *index-based.* This intuition has been finally proposed as a taxonomy by Elnozahy, Alvisi, Wang and Johnson in [7]. The motivation about the separation was that *"..model-based checkpointing maintains certain checkpoint and communication structure which is provably domino-free, and index-based coordination enforces the consistency between checkpoints with the same index ..."*. According to previous definition, model-based based protocols include $\mathcal{VP}$-accordant protocols while index-based encompasses protocols based on sequence numbers like BCS. Using the previous classification, protocols based on vector times (see Section 6.2) would be placed in the model-based family since they do not try to enforce consistency between checkpoints with the same vector time. However, vector time based protocols do not look for domino-free checkpoint and communication structures as we have shown in Section 6.2.

This problem comes from the fact that the above intuitive separation is not a sharp (and formal) criterion. The equivalence between $\mathcal{VP}$ and $\mathcal{NZC}$ provides a sharp (and formal) separation thanks to the fact that $\mathcal{VP}$ sees all protocols as deriving from the same source.

7 Summary

Since Randell's work ([19]), the problem of building a checkpoint and communication pattern in which each checkpoint can be used to avoid domino effect during a rollback phase has been extensively studied in the last two decades. That operational property corresponds formally to the fact that a checkpoint and communication pattern satisfies the $\mathcal{NZC}$ (No-Z-Cycle) property.

This paper provided a characterization of the $\mathcal{NZC}$ property. The characterization is based on a particular type of Z-cycle, namely "core Z-Cycle", that has to be absent from the checkpoint and communication pattern of a distributed execution in order to guarantee the absence of Z-cycles. This result has been obtained thanks to the introduction of concatenation relations which allow the structure of checkpoint and communication patterns to be expressed in easy way. Based on the characterization, we designed a checkpointing protocol that prevents the formation of suspect core Z-cycles whose structure represents the causal part of any core Z-cycle. The protocol lies on the following basic hypothesis: (i) the usable knowledge at a certain event can not be more than the one included in the causal past of that event, (ii) the execution is asynchronous.

Finally we proposed a taxonomy of communication-induced checkpointing protocols that satisfy constraints of Section 5.1. This taxonomy is based on the notion of $\mathcal{VP}$ (Virtual Precedence) property, introduced in [11], which, in the particular context of checkpointing, has been shown to be equivalent to the $\mathcal{NZC}$ property.

8 Acknowledgments

The authors would like to thank Jean-Michel Helary for many email discussions on successive drafts of the paper. A special thank goes to Michel Raynal and Ravi Prakash for many constructive criticism on the original manuscript. We are indebted with the anonymous referees and Faith Fich whose comments were crucial to improve the correctness and the presentation of the paper.

References

[1] R. Baldoni, J.M. Helary and M. Raynal, Rollback Dependency Trackability: a Minimal Characterization and its Protocol. *Information and Computation*, to appear (a short version entitled "Rollback-Dependency Trackability: Visible Characterizations" appeared in *Proc. 18th ACM Symposium on Principles of Distributed Computing*, 1999).

[2] R. Baldoni, J.M. Helary, A. Mostefaoui and M. Raynal, A Communication-Induced Checkpointing Protocol that Ensures Rollback-Dependency Trackability, *Proc. IEEE Int. Symposium on Fault Tolerant Computing*, 1997, pp. 68-77.

[3] R. Baldoni, F. Quaglia and P. Fornara, An Index-Based Checkpointing Algorithm for Autonomous Distributed Systems, *IEEE Transactions on Parallel and Distributed Systems*, vol.10, no.2, 1999, pp. 181-192.

[4] D. Briatico, A. Ciuffoletti and L. Simoncini, A Distributed Domino-Effect Free Recovery Algorithm, *Proc. IEEE Int. Symposium on Reliability Distributed Software and Database*, 1984, pp. 207-215.

[5] K.M. Chandy and L. Lamport, Distributed Snapshots: Determining Global States of Distributed Systems, *ACM Transactions on Computer Systems*, vol. 3, no. 1, 1985, pp. 63-75.

[6] F. Cristian and F. Jahanian, A Timestamp-Based Checkpointing Protocol for Long-Lived Distributed Computations, *Proc. IEEE Int. Symposium on Reliable Distributed Systems*, 1991, pp. 12-20.

[7] E.N. Elnozahy, L. Alvisi, Y.M. Wang and D.B. Johnson, A Survey of Rollback-Recovery Protocols in Message-Passing Systems, *Technical Report No.CMU-CS-99-148, School of Computer Science, Carnegie Mellon University*, 1999.

[8] C. Fidge, Logical Time in Distributed Computing Systems, *IEEE Computer*, pp. 28–33, August 1991.

[9] J. Fowler and W. Zwaenepoel, Causal Distributed Breakpoints, *Proc. IEEE Int. Conference on Distributed Computing Systems*, 1990, pp. 134-141.

[10] E. Fromentin and M. Raynal, Shared Global States in Distributed Computations, *Journal of Computer and System Sciences*, vol. 55, no. 3, December 1997.

[11] J.M. Helary, A. Mostefaoui, and M. Raynal, Virtual Precedence in Asynchronous Systems: Concepts and Applications, *Proc. 11th Workshop on Distributed Algorithms*, LNCS press, 1997.

[12] J.M. Helary, A. Mostefaoui, R.H.B. Netzer and M. Raynal, Communication-Based Prevention of Useless Checkpoints in Distributed Computations, *Distributed Computing*, vol. 13, no. 1, 1999.

[13] R. Koo and S. Toueg, Checkpointing and Rollback-Recovery for Distributed Systems, *IEEE Transactions on Software Engineering*, vol. 13, no. 1, 1987, pp. 23-31.

[14] L. Lamport, Time, Clocks and the Ordering of Events in a Distributed System, *Communications of the ACM*, vol. 21, no. 7, 1978, pp. 558-565.

[15] H.V. Leong and D. Agrawal, Using Message Semantics to Reduce Rollback in Optimistic Message Logging Recovery Schemes, *Proc. IEEE Int. Conference on Distributed Computing Systems*, 1994, pp. 227-234.

[16] D. Manivannan and M. Singhal, A Low-Overhead Recovery Technique Using Quasi Synchronous Checkpointing, *Proc. IEEE Int. Conference on Distributed Computing Systems*, 1996, pp. 100-107.

[17] F. Mattern, Virtual Time and Global States of Distributed Systems, In *Proc. of the International Workshop on Parallel and Distributed Algorithms*, 1989, pp. 215–226.

[18] R.H.B.Netzer and J.Xu, Necessary and sufficient conditions for Consistent Global Snapshots, *IEEE Transactions on Parallel and Distributed Systems*, vol.6, no.2, 1995, pp.165-169.

[19] B. Randell, System Structure for Software Fault Tolerance, *IEEE Transactions on Software Engineering*, vol. SE1, no. 2, 1975, pp. 220-232.

[20] D.L. Russell, State Restoration in Systems of Communicating Processes, *IEEE Transactions on Software Engineering*, Vol. SE6, No. 2, 1980, pp. 183-194.

[21] J. Tsai, Y.M. Wang and S.Y. Kup, Theoretical Analysis of Communication-Induced Checkpointing Protocols with Rollback-Dependency Trackability, *IEEE Transactions on Parallel and Distributed System*, vol. 9, no. 10, 1998, pp. 963-971.

[22] K. Vankatesh, T. Radakrishanan, and H.L. Li. Optimal Checkpointing and Local Recording for Domino-Free Rollback-Recovery, *Information Processing Letters*, vol. 25, 1987, pp. 295-303.

[23] Y.M. Wang, Consistent Global Checkpoints that Contain a Given Set of Local Checkpoints, *IEEE Transactions on Computers*, vol. 46, no. 4, 1997, pp. 456-468.

[24] Y.M. Wang and W.K. Fuchs, Scheduling Message Processing for Reducing Rollback Propagation, *Proc. IEEE Int. Symposium on Fault Tolerant Computing*, 1992, pp. 204-211.