HAL
archives-ouvertes.fr

Consistent Checkpointing in Message Passing Distributed Systems

Roberto Baldoni, Jean-Michel Hélary, Achour Mostefaoui, Michel Raynal

▶ **To cite this version:**

Roberto Baldoni, Jean-Michel Hélary, Achour Mostefaoui, Michel Raynal. Consistent Checkpointing in Message Passing Distributed Systems. [Research Report] RR-2564, INRIA. 1995. <inria-00074117>

HAL Id: inria-00074117
https://hal.inria.fr/inria-00074117

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Consistent Checkpointing in Message Passing Distributed Systems

Roberto Baldoni, Jean Michel Helary

Achour Mostefaoui , Michel Raynal

N˚ 2564

Juin 1995

PROGRAMME 1

ISSN 0249-6399

Consistent Checkpointing in Message Passing Distributed Systems *

Roberto Baldoni**, Jean Michel Helary **
Achour Mostefaoui **, Michel Raynal **

Programme 1 — Architectures parallèles, bases de données, réseaux et systèmes distribués
Projets Adp

Rapport de recherche n ° 2564 — Juin 1995 — 25 pages

Abstract: A global checkpoint of a distributed computation is a a set of local checkpoints (local states), one per process. Determining consistent global checkpoints is an important problem for many distributed applications (e.g. fault-tolerance, distributed debugging, properties detection, etc). This paper focuses on such determinations. A precedence relation on checkpoint intervals (such intervals are sets of events produced by processes between two successive local checkpoints) is introduced and analyzed. It is shown that a local checkpoint is useless (i.e. it cannot participate in any consistent global checkpoint) iff some pattern occurs in this precedence relation. Then an adaptive checkpointing algorithm is introduced. This algorithm, assuming processes take local checkpoints independently, requires them to take (as few as possible) additional checkpoints in order that none of local checkpoints be useless. It is based on the prevention of the previously mentioned pattern. In some sense, this algorithm combines advantages of both coordinated and uncoordinated checkpointing algorithms without inheriting their drawbacks.

Key-words: Consistent Global Checkpoints, Message Communication Systems, Adaptive Checkpointing, Causality, Happened-before Relation.

(Résumé : tsvp)

*Work partially supported by the following Basic Research Action Programs of the European Community: the HCM project under contract No.3702 "CABERNET" and the ESPRIT project under conder contract No. 6360 "BROADCAST".

**IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France. Roberto Baldoni is on leaving from Dipartimento di Informatica e Sistemistica, University of Rome "La Sapienza", via Salaria 113, I-00198 Rome, Italy. This author is also supported by a grant of the *Consiglio Nazionale delle Ricerche* under contract No.93.02294.CT12

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 84 71 71

Les points de contrôle dans les systèmes répartis

Résumé : Dans une exécution répartie, un point de contrôle global est un ensemble de points de contrôle locaux (états locaux), un par processus participant à l'exécution. La détermination d'un point de contrôle global cohérent est un problème important dans de nombreux domaines concernés par les applications réparties (résistance aux défaillances, mise au point répartie, détection de propriétés, etc.). Cet article est consacré à la capture de tels ensembles cohérents.

Une relation de précédence sur les intervalles est introduite (un intervalle est l'ensemble des évènements produits par un processus entre deux points de contrôle locaux consécutifs). On montre qu'un point de contrôle local est inutile (c'est-à-dire ne peut appartenir à aucun point de contrôle global cohérent) si et seulement si un certain motif apparaît dans cette relation.

Puis un algorithme adaptatif de calcul de point de contrôle est construit. Sachant que des points de contrôle locaux peuvent être pris indépendamment par les processus, ceux-ci peuvent être forcés à prendre des points de contrôle supplémentaires afin de maintenir invariante la propriété suivante : *aucun point de contrôle local n'est inutile* (le principe consiste à éviter l'occurrence du motif sus-mentionné). De plus, le nombre de points de contrôle locaux forcés est aussi faible que possible. Enfin à chaque point de contrôle local est associé, dynamiquement, un vecteur local définissant un point de contrôle global cohérent. En ce sens, cet algorithme combine les avantages des algorithmes coordonnés et non-coordonnés, sans en avoir les inconvénients.

Mots-clé : Points de contrôle cohérents, Algorithme adaptatif, Causalité, Communication par messages, Précédence causale.

Contents

1 Introduction

The notion of consistent global checkpoint (also referred to as consistent global state) is fundamental to many important areas of distributed systems such as parallel and distributed debugging ([3], [11]), distributed computing ([1], [5], [7]), fault-tolerance ([6], [13], [14], [16], [17]), detection of stable properties ([2], [4]), etc. A *local checkpoint* is a local state of process and a *global checkpoint* (or global state) is a set of local checkpoints, one from each process constituting the distributed computation. A global checkpoint is consistent if, for all its local checkpoints, no one *happens before* ([9]) another, i.e. there are no messages (or sequence of messages) sent after a local checkpoint and received before another one. Informally, a global checkpoint is consistent if the computation might have passed through it. Formalizations of the notion of consistent global checkpoints can be found in [2], [8] and [12].

Many algorithms have been proposed to determine consistent global checkpoints. Basically each process P_i is associated with a controller CT_i that selects some local states of P_i as being local checkpoints. These checkpointing algorithms can be divided into two classes according to the way local checkpoints are determined to constitute consistent global checkpoints.

- In the class of *coordinated* algorithms, determination of local checkpoints by controllers is synchronized in such a way that the resulting global checkpoint is guaranteed to be consistent ([2], [6]).
- In the class of *uncoordinated* algorithms, each controller CT_i determines, whenever it wants, local checkpoints of process P_i. Then, when a consistent global checkpoint is required, it has to be constructed from the available set of local checkpoints. This task is achieved by an additional protocol, that can be centralized if local checkpoints are sent to a global manager, or distributed if each controller keeps its local checkpoints. The periodic checkpointing algorithm and Russell's algorithm ([14]) are well-known examples of such a class.

Algorithms of the first class guarantee, *a priori*, that all the local checkpoints will be *useful* (i.e. any local checkpoint taken by a controller will be member of a consistent global checkpoint). This is obtained by a cooperation between controllers: when they take checkpoints they synchronize their actions; such a cooperation involves additional control messages and delays. Algorithms of the second class avoid this synchronization but, when a consistent global checkpoint is required, it has to be built by piecing together local checkpoints previously taken. If some local checkpoints have not been taken (or equivalently messages have not been logged) there is no certainty that a consistent global checkpoint can be actually built; in fact, some local checkpoints may be *useless* in the sense that they cannot belong to any consistent global checkpoint (in the context of fault-tolerant systems, based on backward recovery, this drawback is called *domino effect* ([13])).

In this paper we are interested in determining consistent global checkpoints of distributed computations by using an *adaptive* method. In such an approach, controllers may take or not local checkpoints in an arbitrary way by using an uncoordinated checkpointing algorithm (in fact, in a real use, these algorithms depend on the aim of the checkpointing: recovery, detection of properties, etc). In consequence, according to causal dependences on these uncoordinated local checkpoints, some of them can be useless. The adaptive checkpointing algorithm forces then controllers to take (as few as possible) additional local checkpoints in order that all the local checkpoints be useful. The adaptive method presented in this paper is based on a necessary and sufficient condition characterizing useful local checkpoints. This condition is expressed in terms of a precedence relation defined on *checkpoint intervals* (a checkpoint interval is the set of all events issued by a process between two successive local checkpoints): the occurrence of a specific pattern into this relation indicates whether a local checkpoint is useless or not. An adaptive distributed checkpointing algorithm is then designed from this necessary and sufficient condition; it ensures that, the pattern mentioned above will never occur in spite of arbitrarily taken checkpoints by the uncoordinated algorithm.

An important point of the algorithm lies in the fact that it associates, on-the-fly and without delay, with each local checkpoint (taken by the uncoordinated or by the adaptive checkpointing algorithm) a vector of local checkpoint numbers that define a consistent global checkpoint to which the local checkpoint belongs. In that sense, the adaptive algorithm

provides the same advantage as those belonging to the class of coordinated algorithms. However, this algorithm neither uses additional control messages nor adds synchronization to the underlying computation. It only requires messages of the computation to piggyback control information. In that sense, it provides the same advantages as uncoordinated algorithms, while ensuring no local checkpoint is useless. Thus, the proposed algorithm complies exactly with the property characterizing useful checkpoints. To our knowledge, this is the first distributed checkpointing algorithm achieving this goal.

The paper is divided into two main parts. First, Section 2 introduces a formal framework to study consistency of global checkpoints: a precedence relation on checkpoint intervals is defined and the necessary and sufficient condition characterizing useless checkpoints is stated and proved. Then, Section 3 presents the adaptive checkpointing algorithm that requires processes to take additional checkpoints and associates, with each local checkpoint, a consistent global checkpoint to which it belongs; this Section gives also a correctness proof of the algorithm.

2 Consistent Sets of Checkpoints

2.1 Distributed Computations

A distributed computation consists of a finite set P of n processes $\{P_1, P_2, \ldots, P_n\}$ that communicate and synchronize only by exchanging messages. We assume that each ordered pair of processes is connected by an asynchronous reliable and directed logical channel whose transmission delays are unpredictable but finite. Each process runs on a processor and processors do not have a shared memory, there is no bound for their relative speeds and they fail according to the fail-stop model ([15]).

A process can execute internal, send and delivery statements. An internal statement does not involve communication. When P_i executes a send statement *send(m) to* P_j it puts the message m into the channel from P_i to P_j. When P_i executes the statement *delivery(m)*, it is blocked till at least one message directed to P_i has arrived; then a message is withdrawn from one of its input channels and delivered to P_i. Executions of internal, send and delivery statements are modeled by internal, sending and delivery events.

Processes of a distributed computation are *sequential*, in other words, each process produces a *sequence* of events. This sequence of events is called the *history* of P_i, and it is denoted by $h_i = e_i^0 e_i^1 \ldots e_i^s, \ldots$, where e_i^s is s-th event executed by P_i (e_i^0 is a fictitious event that initializes P_i's local state). Let h_i^s denote the *partial history* of P_i till the event e_i^s; $h_i^s = e_i^0 e_i^1 \ldots e_i^s$ is a prefix of h_i.

Events local to a process are totally ordered. However, as each process progresses at its own speed, message transmission delays are unpredictable and there is neither a shared memory nor a global time function, the best that can be known on the the respective event occurrences is only a partial order. Let "$\rightarrow$" denote the *causal precedence* (*happened-before*) partial order defined as follows ([9]).

Definition 2.1 : Relation $\rightarrow$.

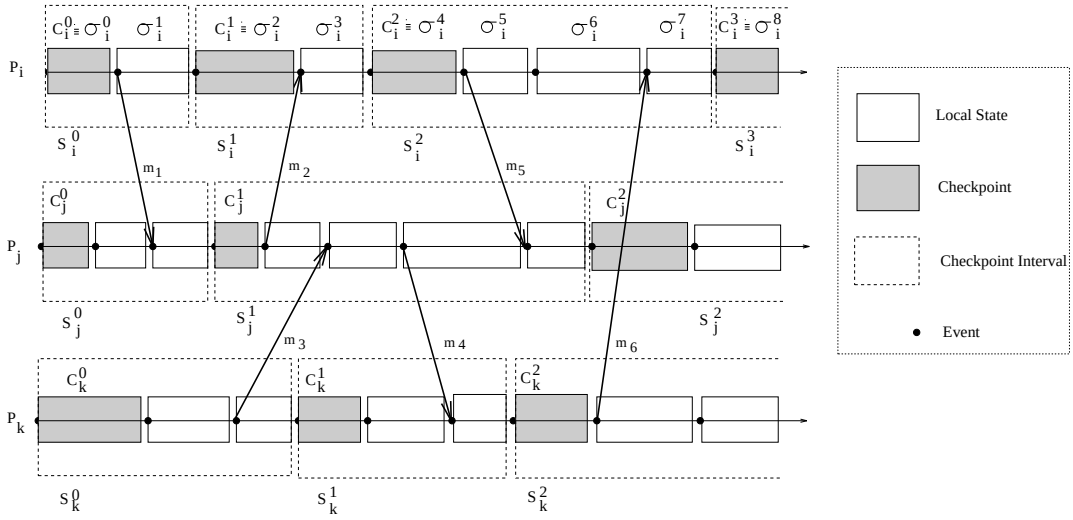


Figure 1: Example of distributed computation

$e_i^s \rightarrow e_j^t$ *if and only if :*

1. $i = j$ *and* $t = s + 1$, *or*
2. $i \neq j$ *and* e_i^s *is the sending event of message* m *and* e_j^t *is the delivery event of* m, *or*
3. *there is an event* e_k^u *such that* $e_i^s \rightarrow e_k^u \wedge e_k^u \rightarrow e_j^t$.

Let H be the set of all the events produced by a distributed computation; this computation is modeled by the partial order $\widehat{H} = (H, \rightarrow)$. Let e be an event $\widehat{H}$; the *causal past* of e is the subset of $\widehat{H}$ including all events e' such that $e' \rightarrow e$.

2.2 Checkpoints

2.2.1 Local and Global Checkpoints

A local state of P_i is a mapping from the set of its local variables identifiers to a set of values. Let σ_i^0 be the initial state of process P_i. The event e_i^s moves P_i from the local state σ_i^{s-1} to the local state σ_i^s. The local state σ_i^s corresponds to the local history h_i^s; by definition e_i^x *belongs to* σ_i^s if $i = j$ and $x \leq s$. Figure 1 shows a distributed computation in the usual space-time diagram where local states of processes are depicted by rectangular boxes.

A *local checkpoint* C is a local state of a process and the set of all the local checkpoints is a subset of all the local states. Whether a local state is or not a local checkpoint does not

depend on the computation itself; checkpoints are determined by controllers of underlying protocols which are superimposed on the computation (We will assume in Section 3 that controllers may or not take some checkpoints following some uncoordinated algorithm, while remaining checkpoints are taken by the adaptive algorithm).

In the following we consider a distributed computation $\widehat{H}$ and a set of local checkpoints $\mathcal{C}_{\widehat{\mathcal{H}}}$ defined on $\widehat{H}$. C_i^x represents the x-th checkpoint taken by process P_i and corresponds to some local state σ_i^s with $x \leq s$. Figure 1 shows a correspondence between local states and checkpoints of process P_i. We assume that each process takes an initial checkpoint C_i^0, corresponding to σ_i^0, when it starts.

A message sent by process P_i to process P_j is called *orphan* with respect to the ordered pair of checkpoints $(C_i^x,\ C_j^y)$ if its delivery event belongs to C_j^y and its sending event does not belong to C_i^x. An ordered pair of checkpoints is *consistent* if and only if there are no orphan messages with respect to this pair. For example, Figure 1 shows the pair (C_k^1,C_j^1) is consistent, while the pair (C_i^2,C_j^2) is inconsistent.

It is easy to see that if a message m is orphan with respect to an ordered pair of checkpoints $(C_i^x,\ C_j^y)$ then, it is also orphan for all the ordered pairs of checkpoints $(C_i^s,\ C_j^t)$ such that $s \leq x$ and $t \geq y$. Thus, if the ordered pair (C_i^x,C_j^y) is not consistent, none of the ordered pair $(C_i^s,\ C_j^t)$ (with $s \leq x$ and $t \geq y$) can be consistent.

A *global checkpoint* is a set of local checkpoints one for each process. For example, $\{C_i^1, C_j^1, C_k^1\}$ and $\{C_i^2, C_j^2, C_k^1\}$ are two global checkpoints depicted in the Figure 1.

Definition 2.2 *A global checkpoint is consistent if all the distinct pairs of checkpoints are consistent.*

For example, Figure 1 shows that $\{C_i^1, C_j^1, C_k^1\}$ is a consistent global checkpoint, while $\{C_i^2, C_j^2, C_k^1\}$ is not consistent.

2.2.2 Checkpoint Intervals

Using an abstraction level defined by local checkpoints, any two successive local checkpoints C_i^x and C_i^{x+1} of process P_i define an interval. We call *checkpoint interval* S_i^x the set of events produced by process P_i between C_i^x and C_i^{x+1} (including the event that produced the local state corresponding to C_i^x). In Figure 1 intervals are depicted as rectangular boxes with dotted lines.

From this definition, it is easy to see that, if a message m is sent in a checkpoint interval S_i^x and delivered in a checkpoint interval S_j^y then m is orphan with respect to the ordered pair of checkpoints (C_i^x, C_j^{y+1}) (and thus with respect to all ordered pairs (C_i^s, C_j^t) with $s \leq x$ and $t \geq y+1$).

2.2.3 Precedence Relation on Checkpoint Intervals

This Section introduces a precedence relation on checkpoint intervals, denoted $\prec$. This relation shows a causal dependence on checkpoint intervals and constitutes the formal framework from which the algorithm of Section 3 is designed.

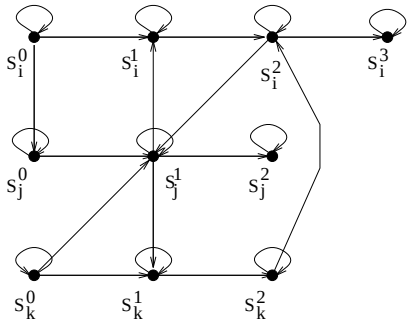


Figure 2: Precedence relation

Definition 2.3 : Relation $\prec$.
S_i^x precedes S_j^y (denoted $S_i^x \prec S_j^y$) if and only if:

1. *$i = j$ and $y = x$, or*
2. *$i = j$ and $y = x + 1$, or*
3. *($\exists$ message m): the sending event of m belongs to S_i^x and the delivery event of m belongs to S_j^y, or*
4. *$\exists z \; \exists k : \; S_i^x \prec S_k^z \; \wedge \; S_k^z \prec S_j^y$.*

Note that the relation $\prec$ is not a partial order (although reflexive and transitive, it is not antisymmetric) and then it can have cycles including distinct checkpoint intervals. As an example, Figure 2 shows the relation $\prec$ of the computation of Figure 1 where, for clarity's sake, we do not consider precedences due to transitivity (point 4. of Definition 2.3).

2.2.4 Sequences of Messages

When $S_i^x \prec S_j^y$ holds, there is a sequence $S_i^x = S_{i_0}^{x_0} \prec S_{i_1}^{x_1} \prec \ldots \prec S_{i_q}^{x_q} = S_j^y$ (if $i \neq j$, then necessarily $q \geq 1$) where each pair $S_{i_k}^{x_k} \prec S_{i_{k+1}}^{x_{k+1}}$ is due to one of points 1., 2., or 3. of Definition 2.3. Due to the possibility of cycles, the elements of this sequence are not necessarily distinct. However, with $q \geq 1$, a subsequence including only distinct intervals, starting at S_i^x and ending at S_j^y, can be extracted; so, we can always consider, without loss of generality, the case where elements of this sequence are distinct (except may be the first and the last one). For each k ($0 \leq k \leq q-1$) we have $S_{i_k}^{x_k} \prec S_{i_{k+1}}^{x_{k+1}}$, where $\prec$ is due to part 2. or 3. of the definition 2.3, i.e.:

- either $i_k = i_{k+1}$ and $s_{k+1} = s_k + 1$

- or $i_k \neq i_{k+1}$ and there is a message m sent in $S_{i_k}^{x_k}$ and delivered in $S_{i_{k+1}}^{x_{k+1}}$

These messages form a sequence $(m_r)_{r=1,\ldots,p}$ (if $i \neq j$ then necessarily $p \geq 1$). Such a sequence of messages, involved in the definition of $S_i^x \prec S_j^y$, is called a winding sequence of messages, abbreviated as *msg-winding*. For example, the msg-winding m_3, m_2 shown on Figure 1 corresponds to the pair $S_k^0 \prec S_i^1$ shown on Figure 2 .

Two types of msg-windings can be distinguished: *causal* and *non-causal* msg-windings.

1. the msg-winding is *causal* if the delivery event of each message of the sequence (except the last) occurs *before* the send event of the next one (sequences m_1, m_2 and m_1, m_4, shown on Figure 1 are examples of causal msg-windings from S_i^0 to S_i^1 and from S_i^0 to S_k^1 respectively).

2. the msg-winding is *non-causal* if at least two consecutive messages are such that the delivery event of the first occurs *after* the send event of the second, although, by definition, these two events belong to the same checkpoint interval (the sequence m_3, m_2 shown on Figure 1 is an example of non-causal msg-winding from S_k^0 to S_i^1).

As usual, the operation of *concatenation* of two msg-windings (μ) and (μ') will be denoted $(\mu) \cdot (\mu')$.

2.3 Characterizing Consistent Sets of Local Checkpoints

This Section states and proves a necessary and sufficient condition to determine whether an arbitrary set of local checkpoints can belong to some consistent global checkpoint. This theorem is based on the precedence relation defined on checkpoint intervals. Another necessary and sufficient condition has been described in [12]; it is expressed in terms of a relation (called *zigzag*) defined on the set of local checkpoints. Both conditions are equivalents from a theoretical point of view. The additional interest of the formulation based on checkpoint intervals lies in the framework it offers to design a distributed adaptive checkpointing algorithm as we shown in Section 3.

Theorem 2.1 *Let $\mathcal{I} \subseteq \{1, \ldots, n\}$ and $\mathcal{LC} = \{C_i^{x_i}\}_{i \in \mathcal{I}}$ be a set checkpoints of $\mathcal{C}_{\widehat{\mathcal{H}}}$. Then $\mathcal{LC}$ is a part of a consistent global checkpoint included in $\mathcal{C}_{\widehat{\mathcal{H}}}$ if and only if:*

$$(\mathcal{P}) \quad \forall i,\ \forall j \ :\ i \in \mathcal{I},\ j \in \mathcal{I} \ ::\ \neg(S_i^{x_i} \prec S_j^{x_j - 1})$$

Proof

Sufficiency. We prove that if $\mathcal{P}$ is satisfied then $\mathcal{LC}$ can be included in a consistent global checkpoint. Let us consider the global checkpoint defined as follows:

- if $i \in \mathcal{I}$, we take $C_i^{x_i}$;
- if $i \notin \mathcal{I}$, for each $j \in \mathcal{I}$ we consider the integer $m_i(j) = min\{y \mid \neg(S_i^y \prec S_j^{x_j - 1})\}$ (with $m_i(j) = 0$ if this set is empty). Then we take $C_i^{x_i}$ with $x_i = \min_{j \in \mathcal{I}}(m_i(j))$. Thus, by definition, $\forall j \in \mathcal{I} \ ::\ \neg(S_i^{x_i} \prec S_j^{x_j - 1})$ and $\exists k \in \mathcal{I} \ ::\ \neg(S_i^{x_i - 1} \prec S_k^{x_k - 1})$.

We show that $\{C_1^{x_1}, C_2^{x_2}, \ldots, C_n^{x_n}\}$ is consistent. Assume the contrary; there exists i and j and a message m such that $send(m) \notin C_i^{x_i}$ and $delivery(m) \in C_j^{x_j}$ (i.e. m is orphan with respect to these local checkpoints) and thus, from point 3. of Definition 2.3, we have:

$$S_i^{x_i} \prec S_j^{x_j-1} \tag{1}$$

Four cases have to be considered:

1. $i \in \mathcal{I}$, $j \in \mathcal{I}$. Relation (1) contradicts the assumption $\neg(S_i^{x_i} \prec S_j^{x_j-1})$;
2. $i \in \mathcal{I}$, $j \notin \mathcal{I}$. By definition of x_j follows that: $\exists k \ : \ k \in \mathcal{I} \ :: \ S_j^{x_j-1} \prec S_k^{x_k-1}$.

   By transitivity (using Relation (1)) we have $S_i^{x_i} \prec S_k^{x_k-1}$ which contradicts the assumption $\mathcal{P}$;
3. $i \notin \mathcal{I}$, $j \in \mathcal{I}$. Relation (1) contradicts the definition of x_i ;
4. $i \notin \mathcal{I}$, $j \notin \mathcal{I}$. By the definition of x_j, $\exists k \ : \ k \in \mathcal{I} \ :: \ S_j^{x_j-1} \prec S_k^{x_k-1}$.

   By transitivity (using Relation (1)) we have $S_i^{x_i} \prec S_k^{x_k-1}$ which contradicts the definition of x_i.

Necessity. We prove that if there is a consistent global checkpoint $\{C_1^{x_1}, C_2^{x_2}, \ldots, C_n^{x_n}\}$ including $\mathcal{LC}$ then property $\mathcal{P}$ holds for any $\mathcal{I} \subseteq \{1, \ldots, n\}$. Assume the contrary: there exist $i \in \mathcal{I}$ and $j \in \mathcal{I}$ such that $S_i^{x_i} \prec S_j^{x_j-1}$. From the definition of Relation $\prec$, there exists a msg-winding $m_1, m_2, \ldots, m_p$ such that :

$$\begin{array}{lll}
 & send(m_1) \in S_i^{x_i}, & \\
delivery(m_1) \in S_{i_1}^{y_1}, & send(m_2) \in S_{i_1}^{z_1} & \text{with } y_1 \leq z_1 \\
 & \ldots & \\
delivery(m_{p-1}) \in S_{i_{p-1}}^{y_{p-1}}, & send(m_p) \in S_{i_{p-1}}^{z_{p-1}} & \text{with } y_{p-1} \leq z_{p-1} \\
delivery(m_p) \in S_j^{x_j-1} & &
\end{array}$$

We show by induction on p that $\forall t \geq x_j$, $C_i^{x_i}$ and C_j^t cannot belong to the same global checkpoint.

Base step. $p = 1$. In this case, $send(m_1) \in S_i^{x_i}$ and $delivery(m_1) \in S_j^{x_j-1}$ thus m_1 is orphan in all ordered pairs $(C_i^{x_i}, C_j^t)$ with $t \geq x_j$.

Induction step. We suppose the result true for some $p \geq 1$ and show that it holds for $p+1$. We have:

$$\begin{array}{ll} & send(m_1) \in S_i^{x_i}, \\ & \ldots \\ delivery(m_p) \in S_{i_p}^{y_p}, & send(m_{p+1}) \in S_{i_p}^{z_p} \quad \text{with } y_p \leq z_p \\ delivery(m_{p+1}) \in S_j^{x_j - 1} & \end{array}$$

From the assumption induction applied to the sequence $m_1, \ldots, m_p$, we have : for any $t \geq y_p + 1$, $C_i^{x_i}$ and $C_{i_p}^t$ cannot belong to the same consistent global checkpoint. Moreover, $send(m_{p+1}) \in S_{i_p}^{z_p}$ and $delivery(m_{p+1}) \in S_j^{x_j - 1}$ imply that, for any $u \leq z_p$ and for any $t \geq x_j$, $C_{i_p}^u$ and C_j^t cannot belong to the same consistent checkpoint. Since $y_p \leq z_p$, it follows that no checkpoint in process P_{i_p} can be included with $C_i^{x_i}$ and $C_j^{x_j}$ to form a global consistent checkpoint. □

This Theorem has also an interesting corollary which allows to decide if a local checkpoint can be a member of any consistent global checkpoint, in other words whether it is *useful* or not.

Corollary 2.2 *A checkpoint C_i^x can belong to some consistent global checkpoint if and only if:*

$$\neg(S_i^x \prec S_i^{x-1})$$

Local checkpoints that verify Corollary 2.2 are called *useful*. A non-useful (or *useless*) checkpoint cannot belong to any consistent global checkpoint. Checkpoints C_i^1, C_j^1 and C_k^1 shown in Figure 1 are examples of useful checkpoints, whereas C_i^2 and C_k^2 are useless. So, the existence of a non useful checkpoint (C_i^x) means that a checkpoint interval (S_i^x) precedes another one (S_i^{x-1}) that belongs to its past. As an example, the checkpoint C_k^2 shown in Figure 1 corresponds to the precedence $S_k^2 \prec S_k^1$, transitive consequence of $S_k^2 \prec S_i^2$, $S_i^2 \prec S_j^1$ and $S_j^1 \prec S_k^1$ (Figure 2). This simple consideration constitutes the basis from which the checkpointing algorithm of the next Section is designed.

3 An Adaptive Checkpointing Algorithm

This Section presents an adaptive checkpointing algorithm. Controllers are supposed to take local checkpoints in an arbitrary way following an uncoordinated algorithm. The adaptive checkpointing algorithm forces controllers to take (as few as possible) additional checkpoints in order that no checkpoint be useless.

The decisions about when and which controllers have to take these additional local checkpoints is based on the relation $\prec$ defined on checkpoint intervals, introduced in Section 2.2.3: there will be no checkpoint interval S_a^α such that $S_a^\alpha \prec S_a^{\alpha-1}$ or, equivalently (as shown in Section 2.3), all the checkpoints will be made useful. The strategy ensuring this property is developed in Sections 3.1 and 3.2.

The algorithm adds no synchronization and no control messages to the computation and uses only the piggybacking of control information on messages of the computation in order

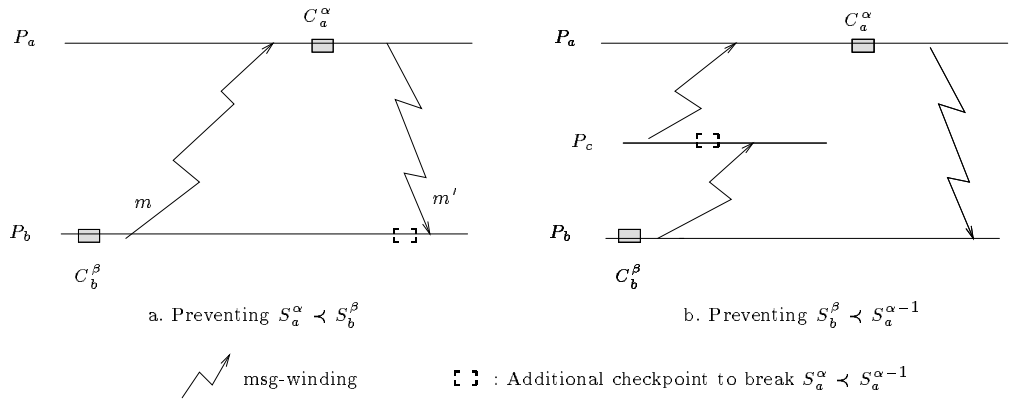


Figure 3: Rewinding paths on C_a^α

to take consistent decisions. Moreover, when a checkpoint is taken by a controller (either independently or forced by the algorithm) it is associated by the algorithm with a vector of checkpoint numbers, one per process, defining a consistent global checkpoint. This important feature is developed in Section 3.4 where the correctness of the algorithm is proved.

For the sake of simplicity and without loss of generality, through this Section, we will merge controllers CT_i with their associated processes P_i .

3.1 Principle of the Algorithm: Avoiding Rewinding Paths

As shown in the previous section, a checkpoint C_a^α is useless if and only if $S_a^\alpha \prec S_a^{\alpha-1}$. But this relation is equivalent to:

$$\exists b \ : b \neq a \ :: \ (S_a^\alpha \prec S_b^\beta) \wedge (S_b^\beta \prec S_a^{\alpha-1})$$

Definition 3.1 : Rewinding Paths.
A path, in relation $\prec$, from S_a^α to $S_a^{\alpha-1}$ is called a rewinding path *on C_a^α.*

In order to make the checkpoint C_a^α useful, the associated rewinding path must be broken. This can be done by breaking either the path $S_a^\alpha \prec S_b^\beta$ or the path $S_b^\beta \prec S_a^{\alpha-1}$.

1. $S_a^\alpha \prec S_b^\beta$ is broken if P_b takes at least one additional checkpoint between the sending of the message m and the delivery of the message m' as described in Figure 3.a: in that case, the interval S_b^β will not include the delivery of m' and thus $S_a^\alpha \prec S_b^\beta$ will not hold.

2. When the msg-winding from S_b^β to $S_a^{\alpha-1}$ is not causal, $S_b^\beta \prec S_a^{\alpha-1}$ is broken by requiring some process involved in this msg-winding to take a checkpoint (see Figure 3.b).

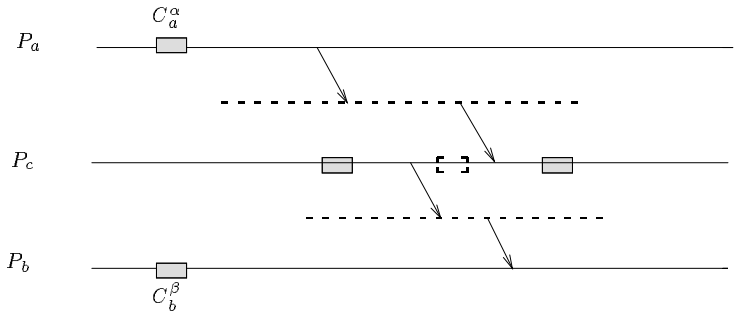


Figure 4: Non-causal msg-winding

These two strategies are discussed in Sections 3.2.1 and 3.2.2, respectively.

3.2 Strategies to Avoid Rewinding Paths

Practically, the strategy will be selected according to the information available at each process. This information is updated by control values piggybacked on computation messages, and depends on the nature of the msg-windings involved in rewinding paths: *causal* or *not causal* (recall that, for each (a, α) and (b, β), when $S_a^\alpha \prec S_b^\beta$ (with $a \neq b$) holds, there is at least one msg-winding from P_a to P_b).

Definition 3.2 : Relation $\prec_\mathcal{C}$.
$S_a^\alpha \prec_\mathcal{C} S_b^\beta$ *if and only if there exists at least one* causal *msg-winding starting in* $S_a^{\alpha'}$ *and finishing in* $S_b^{\beta'}$, *with* $\alpha' \leq \alpha$ *and* $\beta \leq \beta'$.

Clearly, the relation $\prec_\mathcal{C}$ is included in the relation $\prec$, but the converse is not true.

Definition 3.3 : Breakable msg-winding.
A msg-winding is said breakable *by a process* P_c *if it is* non-causal *and if the delivery event of a message occurs on* P_c *after the send event of the next message.*

As an example, the msg-winding depicted on Figure 4 is breakable by P_c. This means that P_c can break the non-causal msg-winding by taking a checkpoint between the send and the delivery event (depicted by a dotted box on Figure 4).

Definition 3.4 *A rewinding path* $S_a^\alpha \prec S_a^{\alpha-1}$ *is* causal *if there exists* b, $b \neq a$, *such that* $S_a^\alpha \prec_\mathcal{C} S_b^\beta$ *and* $S_b^\beta \prec_\mathcal{C} S_a^{\alpha-1}$. *Otherwise, the rewinding path is* non-causal.

Figure 5 describes a causal rewinding path $S_a^\alpha \prec S_a^{\alpha-1}$. Let us remark that the only way for processes to "learn" new information concerning other processes is the receipt of messages. For example, a process P_b can "learn" that another process P_a has reached its

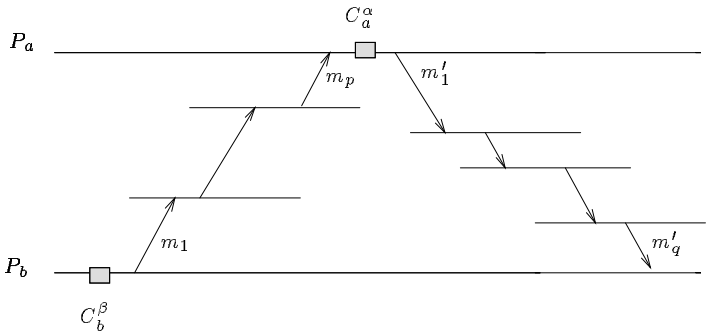


Figure 5: Causal rewinding path

checkpoint interval number S_a^α if there is a causal msg-winding from P_a to P_b, starting in this interval, and if the messages carry the information (a, α). On the other hand, P_b can "learn" that this information is new if the previous knowledge that it had about P_a was an interval number α' such that $\alpha' < \alpha$. In other words, as P_b learns the interval number α of P_a, there exists a causal msg-winding from P_a to P_b, starting in the checkpoint interval S_a^α.

3.2.1 Preventing Causal Rewinding Paths

Such a rewinding path $S_a^\alpha \prec S_a^{\alpha-1}$ is depicted in Figure 5. It can be detected by P_b, thanks to information carried by the message m'_q. This information must indicate to P_b that:

1. m'_q is the last message of a causal msg-winding starting from itself in the same checkpoint interval, namely S_b^β (and so $S_b^\beta \prec_{\mathcal{C}} S_b^\beta$).
2. one of the processes involved in this msg-winding has taken a checkpoint between the delivery of a message and the sending of the next one (P_a in Figure 5).

The first information necessary to detect a causal rewinding path can be piggybacked on messages: each process P_i keeps an array of interval numbers $current\_ckpt_i$, such that, for all j $(1 \leq j \leq n)$, $current\_ckpt_i[j]$ is the current interval number of P_j, to the knowledge of P_i. Clearly, the following invariant holds : $(\forall i)(\forall j) S_j^{current\_ckpt_i[j]} \prec_{\mathcal{C}} S_i^{current\_ckpt_i[i]}$. When P_i sends a message, $current\_ckpt_i$ is piggybacked on the message. Thus, when a message m, sent by P_a, arrives at P_b, this message carries a vector $current$ equal to the value of $current\_ckpt_a$ when m was sent. In particular, $current[b]$ is the current interval number of P_b to the knowledge of P_a when the message m has been sent. Consequently, this message completes a path in the relation $\prec_{\mathcal{C}}$ starting from and returning to $S_b^{current\_ckpt_b[b]}$ if and only if $current[b] = current\_ckpt_b[b]$ (in this case we have $S_b^\beta \prec_{\mathcal{C}} S_b^\beta$ with $\beta = current\_ckpt_b[b]$). When m is delivered, each entry $current\_ckpt_b[j]$ is updated to

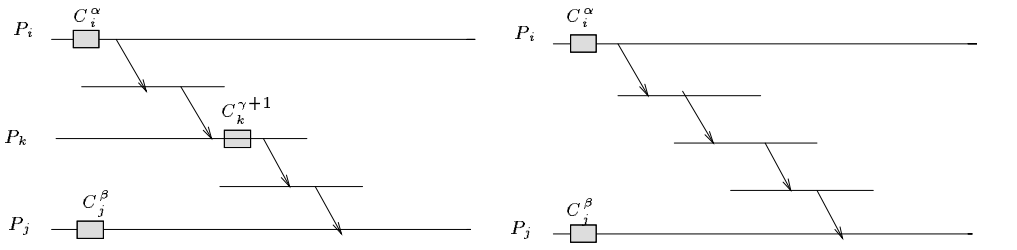


Figure 6: Causal msg-windings

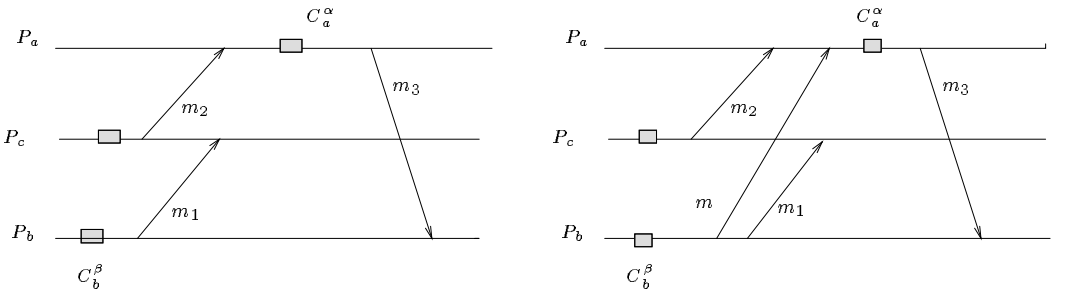
$\max(current\_ckpt_b[j], current[j])$, the updated current interval number of P_j to the knowledge of P_b.

The second information necessary to detect causal rewinding paths will also be piggybacked on messages. Let us remark that each causal msg-winding may include or not intermediate checkpoints, as shown on Figure 6. More precisely, we will say that $S_i^\alpha \prec_{\mathcal{C}} S_j^\beta$ is *compound* if and only if there exists a pair (k, γ) such that $S_i^\alpha \prec_{\mathcal{C}} S_k^\gamma \wedge S_k^{\gamma+1} \prec_{\mathcal{C}} S_j^\beta$ (Figure 6.a). If $S_i^\alpha \prec_{\mathcal{C}} S_j^\beta$ is not compound, it is *simple* (Figure 6.b). But, the path $S_b^\beta \prec_{\mathcal{C}} S_b^\beta$ is a causal rewinding path if and only if $\exists a \ : \ a \neq b \ :: \ (S_b^\beta \prec_{\mathcal{C}} S_a^{\alpha-1}) \wedge (S_a^\alpha \prec_{\mathcal{C}} S_b^\beta)$. Thus, P_b has to determine whether $S_b^\beta \prec_{\mathcal{C}} S_b^\beta$ is compound or not. At this end, each process P_i keeps an array $only\_simple_i$ of booleans, such that, for all j $(1 \leq j \leq n)$, $only\_simple_i[j]$ is true if, to the knowledge of P_i, the relation $S_j^{current\_ckpt_i[j]} \prec_{\mathcal{C}} S_i^{current\_ckpt_i[i]}$ is simple. In other words, up to that point, none of the causal msg-windings issued in $S_j^{current\_ckpt_i[j]}$ and finishing in $S_i^{current\_ckpt_i[i]}$ includes intermediate checkpoints. The consistency of $only\_simple_i$ is maintained by P_i as follows: when P_i takes a checkpoint, it resets all the entries $only\_simple_i[j]$ (with $i \neq j$) to *false*, $only\_simple_i[i]$ remaining *true*. When it sends a message, P_i appends $only\_simple_i$ to it. Thus, when a message m, sent by P_c, arrives at P_b, this message carries the vector *simple* equal to the value of $only\_simple_c$ when m was sent. However, by definition, the value of $only\_simple_c[b]$ is *false* if and only if at least one of the causal msg-windings issued in S_b^β and arrived at P_c includes a checkpoint. As a consequence, the relation $S_b^\beta \prec_{\mathcal{C}} S_b^\beta$ completed by the arrival of m (and detected by the validity of $current[b] = current\_ckpt_b[b]$) is compound if and only if the value of $simple[b]$ is *false*. The array *simple* is also used to update $only\_simple_b$ in order to maintain it correct.

To summarize, when a message $(m, current, simple)$ arrives at P_b, this process checks the condition

$$\mathcal{C}_1 \equiv ((current[b] = current\_ckpt_b[b]) \wedge \neg simple[b])$$

If this condition holds, then the message m completes a causal rewinding path. The algorithm will force P_b to take a checkpoint before delivering the message m.



a. A non-causal rewinding path.

b. A message doubling a non-causal msg-winding.

Figure 7: Non-causal rewinding path

3.2.2 Preventing Non-causal Rewinding Paths

The strategy used in the previous Section was a "last opportunity strategy" in the sense that a causal rewinding path is prevented just before its completion. Indeed, forcing the process P_b to take a checkpoint was the last chance to prevent it. This strategy works because it relies only on the knowledge by P_b of events belonging to its causal past. In the case of non-causal rewinding paths, the previous strategy does not work. Figure 7 depicts a simple case with only three messages; in that example, $\neg(S_b^\beta \prec_{\mathcal{C}} S_a^\alpha)$ and thus, the entry $current[b]$ carried by the message m_3 is such that $current[b] < \beta$ (the up-to-date interval number β for P_b was not known by the intermediate process P_c when it sent m_2). When P_b receives the message m_3, the condition $\mathcal{C}_1$ will be evaluated to *false*, and thus P_b will not break the rewinding path $(S_b^\beta \prec S_a^{\alpha-1}) \wedge (S_a^\alpha \prec S_b^\beta)$. Thus, the strategy here consists in breaking the non-causal msg-winding m_1, m_2 by forcing P_c to take a checkpoint before delivering m_1 (unless the information available at P_c ensures that such a checkpoint will be redundant). This strategy is "conservative" in the following sense: a rewinding path involving m_1 and m_2 could be completed in the future by some exchanges of messages (like m_3) and some checkpointing (like C_a^α) and, to its current knowledge, P_c is the only process that can break it.

A naive solution would consist in forcing a process to take a checkpoint before each message delivery. Although simple, this solution is likely to force processes to take redundant checkpoints. This solution can easily be improved: checkpoints are forced only upon the first message arrival after each send event ([14]). However, both rules (being purely syntactics) don't take advantage of information that can be carried by messages in order to detect redundancy, as shown by the example shown in Figure 7.b. If P_c learns, before the arrival of the message m_1, that $S_b^\beta \prec_{\mathcal{C}} S_a^{\alpha-1}$ (due to the message m in this example) then P_c doesn't need to take a checkpoint before the delivery of m_1. In fact, to the knowledge of P_c upon m_1 arrival, the causal msg-winding (reduced to the message m) "doubles" the non-causal msg-winding m_1, m_2; thus, as seen in Section 3.2.1, the causal cycle will be broken by P_b (which will be forced to take a checkpoint before delivering m_3).

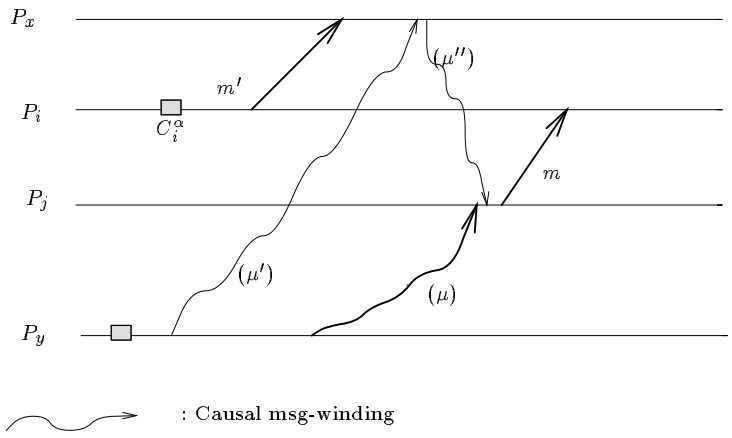


Figure 8: Non-causal msg-winding doubled by a causal one

This simple example suggests that, when a non-causal msg-winding is doubled by a causal one, it can be no longer considered as able to create a non causal rewinding path.

Definition 3.5 : Causal Doubling.
A non causal msg-winding from S_a^α to S_b^β is causally doubled *if there exists a causal msg-winding from S_a^α to S_b^β.*

Note that these two msg-windings do not necessarily involve the same intermediate processes. In the situation depicted in Figure 8, the non causal msg-winding $(\mu) \cdot m \cdot m'$ is causally doubled by (μ').

Consider the situation where a message m arrives at process P_i. This message forms non-causal msg-windings with all messages sent by P_i between its last checkpoint C_i^α and the arrival of m (Figure 8). If P_i decides to take a checkpoint before the delivery of m, it breaks all such non-causal msg-windings. On the contrary, if P_i does not take a checkpoint before the delivery of m, none of these msg-windings is broken by P_i. Thus, if, to the knowledge of P_i, *at least one* of these msg-windings is not causally doubled, the algorithm will force P_i to take a checkpoint before the delivery of m, to prevent the possible formation of a non-causal rewinding path including this non-causal msg-winding. This knowledge requires P_i to fix the following points:

i) Find all msg-windings including m and that P_i can break.

ii) Find whether all these msg-windings are causally doubled or not.

Answering point (**i**) requires to answer the following questions concerning non causal msg-windings: where do they come from, where do they arrive?

1. The answer to the first question lies on knowledge from the causal past of the delivery event of m, and this knowledge is included in the array *current* carried by the message m. In fact, for each y, either $current[y] > current\_ckpt_i[y]$ or not.

   (a) In the first case, P_i learns the new interval number $current[y]$ about P_y, meaning that there is a causal msg-winding from P_y to P_i, starting in the interval $S_y^{current[y]}$, and this msg-winding is the *first* one bringing this information to P_i.

   (b) In the other case ($current[y] \leq current\_ckpt_i[y]$), P_i has previously received a message ending a causal msg-winding issued from $S_y^{current[y]}$. Upon the arrival of this message, P_i decided or not to take a checkpoint, according to its knowledge at that time, and thus, the arrival of m will not change anything to the previous decision.

2. The answer to the second question involves some knowledge on the "future" of the events $send(m')$. But the only information available at P_i when m arrives is the identity of the processes to which P_i has sent messages in its current interval. At this end, each process P_i keeps an array of booleans $sent\_to_i$ such that, for all j ($1 \leq j \leq n$), $sent\_to_i[j]$ is *true* if and only if P_i has sent a message to P_j since its last checkpoint.

The set of non-causal msg-windings breakable by P_i is thus determined by the set of pairs (P_y, P_x) such that $(current[y] > current\_ckpt_i[y]) \wedge sent\_to_i[x]$.

Answering point (**ii**) requires to check whether a non-causal msg-winding is causally doubled. At this end, P_i must have information such that "given two processes P_y and P_x, is there a causal msg-winding from P_y to P_x, starting in the last interval number of P_y known by P_i?". To answer the latter question, each process P_i keeps a boolean matrix $causal\_winding_i$, such that, for all (y, x) ($1 \leq y, x \leq n$), $causal\_winding_i[y, x]$ is true if and only if, to the knowledge of P_i, there is a causal msg-winding from P_y to P_x, starting in the last checkpoint interval of P_y known by P_i (namely $S_y^{current\_ckpt_i[y]}$) and arrived at P_x. When P_i sends a message, $causal\_winding_i$ is piggybacked on the message. Thus, when a message m, sent by P_j, arrives at P_i, this message carries a matrix *causal* whose value is equal to the value of $causal\_winding_j$ when m was sent. The matrix *causal* will also be used to update the value of $causal\_winding_i$ in order to maintain its correction. The situation depicted Figure 8 shows that the existence of the causal msg-winding (μ') doubling the non-causal msg-winding $(\mu) \cdot m \cdot m'$ is known by P_j (thanks to (μ'')) upon the sending of m. Thus, the entry $causal[y, x]$ carried by the message m has the value $true$.

The following condition $\mathcal{C}_2$ summarizes the previous discussion; in disjunction with the condition $\mathcal{C}_1$, it must also be checked by P_i upon the arrival of a message $(m, current, simple, causal)$. If it is satisfied, then the algorithm forces P_i to take a checkpoint before the delivery of m :

$$\mathcal{C}_2 \equiv (\exists x) \ : (sent\_to_i[x] \wedge ((\exists y) \ : \ ((current[y] > current\_ckpt_i[y]) \wedge \neg causal[y, x])))$$

When true, this condition means that, to the knowledge of P_i, there exists a non-causal msg-winding from P_y to P_x, breakable by P_i and not causally doubled.

3.3 Formal Description of the Algorithm

Each process P_i is endowed with the following arrays whose semantics has been defined in the previous Sections.

$current\_ckpt_i$: **array**$[1..n]$ **of** *integer*;
$only\_simple_i$: **array**$[1..n]$**of** *boolean*;
$sent\_to_i$: **array**$[1..n]$**of** *boolean*;
$causal\_winding_i$: **array**$[1..n, 1..n]$ **of** *boolean*;

The algorithm is formally described by the statements performed by a process P_i at initialization (S0), when it takes a new checkpoint (S1), when it sends a message (S2), and when a message arrives (S3). The following procedure describes actions executed by process P_i each time it takes a checkpoint:

procedure take_checkpoint **is**
  $current\_ckpt_i[i] := current\_ckpt_i[i] + 1$;
  $\forall k$ **do** $sent\_to_i[k] := false$ **enddo**;
  $\forall j \neq i$ **do** $only\_simple_i[j] := false$ **enddo** ;
  $\forall j \neq i$ **do** $causal\_winding_i[i, j] := false$ **enddo**;
  store the current local state and a copy of the array $current\_ckpt_i$ on stable storage;
end

(S0) **initialization**
  $\forall k$ **do** $current\_ckpt_i[k] := 0$ **enddo**;
  $only\_simple_i[i] := true$; % actually variables $only\_simple_i[i]$ and %
  $causal\_winding_i[i, i] := true$; % $causal\_winding_i[i, i]$ will remain always $true$ %
  take_checkpoint;
end

(S1) **when**P_i **takes a checkpoint**
  take_checkpoint;
end

(S2) **when** P_i **sends a message to** P_j
  $sent\_to_i[j] := true$;
  send$(m, current_ckpt_i, only_simple_i, causal_winding_i)$;
end

(S3) **when a message** $(m, current, simple, causal)$ **arrives to** P_i **from** P_j

```
        if C1 ∨ C2
          where C1 ≡ (current_ckpt_i[i] = current[i]) ∧ ¬simple[i],
                C2 ≡ ∃x : (sent_to_i[x] ∧
                           ∃y : ((current[y] > current_ckpt_i[y]) ∧ ¬causal[y, x])
                           )
          then take_checkpoint
        endif;
        % updating of control variables %
        ∀k do
          case
            current[k] < current_ckpt_i[k] → skip
            current[k] > current_ckpt_i[k] →
                current_ckpt_i[k] := current[k] ;
                only_simple_i[k] := simple[k] ;
                ∀ℓ do causal_winding_i[k, ℓ] := causal[k, ℓ] enddo;
            current[k] = current_ckpt_i[k] →
                only_simple_i[k] := only_simple_i[k] ∧ simple[k];
                ∀ℓ do causal_winding_i[k, ℓ] := causal_winding_i[k, ℓ] ∨ causal[k, ℓ] enddo;
          endcase
        enddo ;
        causal_winding_i[j, i] := true;
        ∀ℓ do causal_winding_i[ℓ, i] := causal_winding_i[ℓ, i] ∨ causal_winding_i[ℓ, j] enddo;
        deliver(m);
end
```

3.4 Proof of the Algorithm

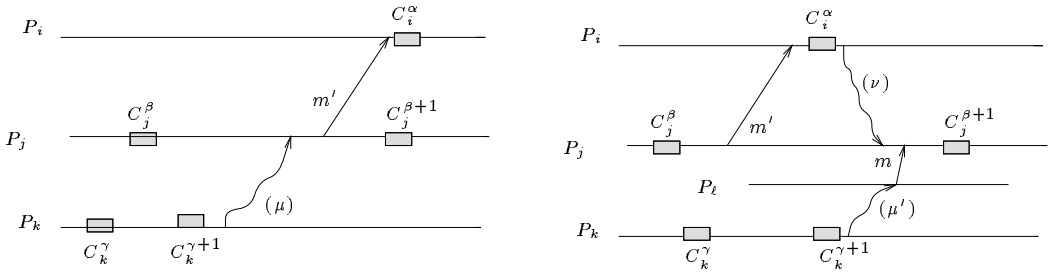
To prove the correctness of the algorithm, we show that no checkpoint is useless. To that end, we will show that each checkpoint C_i^α belongs to a consistent global checkpoint included in $\mathcal{C}_{\widehat{\mathcal{H}}}$ the set of all the local checkpoints. Moreover, this consistent global checkpoint is known locally by P_i: it is easily obtained from the local vector $current\_ckpt_i$.

Lemma 3.1 *Let P_i, P_j, P_k be three processes and α, β, γ be three checkpoint interval numbers, such that (Figure 9):*

1. *there is a message m' from S_j^β to $S_i^{\alpha-1}$.*
2. *$S_k^{\gamma+1} \prec_{\mathcal{C}} S_j^\beta$.*

Then $S_k^{\gamma+1} \prec_{\mathcal{C}} S_i^{\alpha-1}$.

Proof From assumption 2., there is at least a causal msg-winding starting in $S_k^{\gamma+1}$ and arriving at P_j in the checkpoint interval numbered β' with $\beta' \leq \beta$. Let (μ) be the *first* causal msg-winding starting from $S_k^{\gamma+1}$ and arriving at P_j. Two cases have to be considered.



a. $(\mu) \cdot m'$ is a causal msg-winding

b. $(\mu') \cdot m \cdot m'$ is a non-causal msg-winding

Figure 9: The two cases of Lemma 3.1

Case (i): the msg-winding $(\mu) \cdot m'$ is causal. In that case, by definition, $S_k^{\gamma+1} \prec_{\mathcal{C}} S_i^{\alpha-1}$ (Figure 9.a).

Case (ii): the msg-winding $(\mu) \cdot m'$ is non-causal (it is breakable by P_j). Let (μ') be a msg-winding and m be a message such that $(\mu') \cdot m = (\mu)$ (Figure 9.b). The value $current[k]$ carried by m satisfies, by the definition of (μ):

(1) $\quad current[k] = \gamma + 1 > current\_ckpt_j[k]$

Moreover, upon the arrival of m, we have, because of the message m':

(2) $\quad sent\_to_j[i] = true$

Assumptions 1. and 2. show that P_j has not taken any checkpoint between the sending of m' and the arrival of m. In particular, it has not been forced by the algorithm to take a checkpoint upon the arrival of m and thus, both conditions $\mathcal{C}_1$ or $\mathcal{C}_2$ are evaluated to *false* when this message arrives. Consequently:

ii.1. As $\mathcal{C}_1$ is false, it cannot exist any causal msg-winding (ν) starting in S_i^α and ending with m (or before its arrival) in S_j^β. If such a msg-winding existed, the msg-winding $m' \cdot (\nu)$ would have created a causal rewinding path $(S_j^\beta \prec_{\mathcal{C}} S_i^{\alpha-1}) \wedge (S_i^\alpha \prec_{\mathcal{C}} S_j^\beta)$ and hence, due to $\mathcal{C}_1$, P_j would have been forced to take a checkpoint before delivering the last message of (ν).

ii.2. As $\mathcal{C}_2$ is false, we have (*current* and *causal* are the values carried by m):

$$\neg\mathcal{C}_2 \equiv (\forall x)(\neg sent\_to_j[x] \vee ((\forall y)((current[y] \leq current\_ckpt_j[y]) \vee causal[y,x])))$$

With $x = i$ and $y = k$ we obtain:

$$\neg sent\_to_j[i] \vee (current[k] \leq current\_ckpt_j[k]) \vee causal[k,i]$$

From (1) and (2), this reduces to $causal[k,i]$. Let P_ℓ be the sender of m. To the knowledge of P_ℓ, there is a causal msg-winding from P_k to P_i, starting in $S_k^{current\_ckpt_\ell[k]} = S_k^{\gamma+1}$, and finishing in some $S_i^{\alpha'}$; thus, to the knowledge of P_ℓ, $S_k^{\gamma+1} \prec_{\mathcal{C}} S_i^{\alpha'}$. We claim

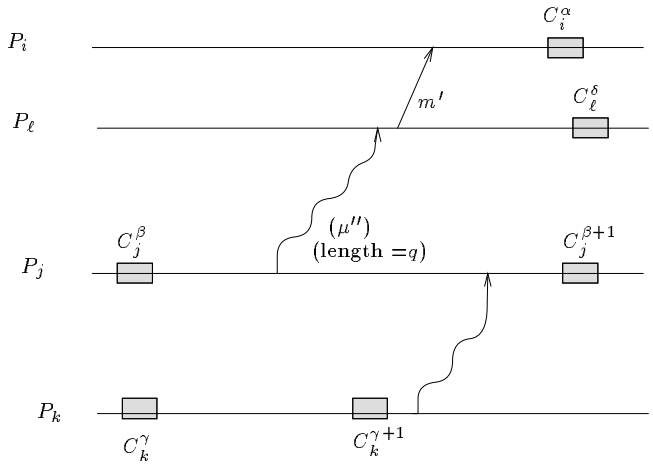


Figure 10: Induction of Lemma 3.2

that $\alpha' < \alpha$. In fact, the relation $S_k^{\gamma+1} \prec_{\mathcal{C}} S_i^{\alpha'}$ is known by P_ℓ when it sends m, and thus there must be a causal msg-winding starting in $S_i^{\alpha'}$ and arriving at P_l before the sending of m. This causal msg-winding, concatenated with m, implies that $S_i^{\alpha'} \prec_{\mathcal{C}} S_j^{\beta}$, and case (ii.1.) shows that this is not possible with $\alpha' \geq \alpha$. Thus, $S_k^{\gamma+1} \prec_{\mathcal{C}} S_i^{\alpha'}$ with $\alpha' < \alpha$, and consequently we have $S_k^{\gamma+1} \prec_{\mathcal{C}} S_i^{\alpha}$.

□

Lemma 3.2 *Let P_i, P_j, P_k be three processes and α, β, γ be three checkpoint interval numbers, such that:*

1. $S_j^{\beta} \prec_{\mathcal{C}} S_i^{\alpha-1}$.
2. $S_k^{\gamma+1} \prec_{\mathcal{C}} S_j^{\beta}$.

Then $S_k^{\gamma+1} \prec_{\mathcal{C}} S_i^{\alpha-1}$.

Proof From assumption 1., there is a causal msg-winding (μ') starting from S_j^{β} and arriving in $S_i^{\alpha-1}$. We proof the result by induction on the length q of the msg-winding (μ').

- $q = 1$. The msg-winding (μ') is reduced to a message m' and the result follows from lemma 3.1.
- Suppose the result true for all msg-windings of length q. We show that it remains true for a msg-winding of length $q + 1$. Such a msg-winding is the concatenation of

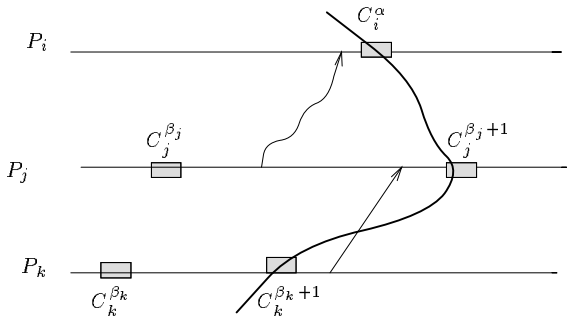


Figure 11: Consistency of a global checkpoint

a msg-winding (μ'') of length q, from S_j^{β} to $S_\ell^{\delta-1}$ and a message m' from $S_\ell^{\delta-1}$ to $S_i^{\alpha-1}$ (Figure 10). From the induction assumption, we have: $S_k^{\gamma+1} \prec_{\mathcal{C}} S_\ell^{\delta-1}$, and thus assumption 1. of Lemma 3.1 holds. The assumption 2. of Lemma 3.2 is satisfied by the message m'. Thus the required relation $S_k^{\gamma+1} \prec_{\mathcal{C}} S_i^{\alpha-1}$ holds.

□

Theorem 3.3 *Let P_i be a process, α be a checkpoint number of this process, and $CKPT_i^{\alpha}$ denote the vector associated with C_i^{α} and defined as follows:*

- $CKPT_i^{\alpha}[i] = \alpha$
- $\forall j : j \neq i :: CKPT_i^{\alpha}[j] = \beta_j + 1$ *where β_j is the value of $current\_ckpt_i[j]$ when P_i took its checkpoint C_i^{α}.*

Then the vector $CKPT_i^{\alpha}$ defines a consistent global checkpoint.

Proof We show that there cannot exist any message from a process P_k to a process P_j, received in $S_j^{\beta_j}$ and sent after $S_k^{\beta_k+1}$ (i.e., no messages is orphan with respect to the ordered pair $(C_k^{\beta_k+1}, C_j^{\beta_j+1})$).

Suppose that such a message exists (see Figure 11, where the line going through $C_i^{\alpha}, C_j^{\beta_j+1}, C_k^{\beta_k+1}$ represents a global checkpoint). Due to the definition of β_j, we have $S_j^{\beta_j} \prec_{\mathcal{C}} S_i^{\alpha-1}$. By the assumption, we have $S_k^{\beta_k+1} \prec_{\mathcal{C}} S_j^{\beta}$. Thus, from Lemma 3.2, $S_k^{\beta_k+1} \prec_{\mathcal{C}} S_i^{\alpha-1}$. From this it follows that $current_i[k] \geq \beta_k + 1$ when P_i took its checkpoint C_i^{α}. This contradicts the definition of β_k. □

4 Conclusion

The notion of consistent global checkpoint is fundamental in many research areas in distributed systems. In this paper a formalization of this notion has been introduced and a distributed adaptive algorithm is designed. Both of them lie on a precedence relation on checkpoint intervals.

In the first part of the paper, a necessary and sufficient condition to determine whether an arbitrary set of local checkpoints can belong to some consistent global checkpoint has been stated and proved. An important consequence of this theorem shows that the occurrence of a specific pattern in the precedence relation is equivalent to the existence of a local checkpoint that cannot belong to any consistent global checkpoint (i.e., useless checkpoint). Informally, this pattern occurs if, on the same process, a checkpoint interval precedes another checkpoint interval that belongs to its past. From a theoretical viewpoint this analysis is equivalent to the one introduced in [12] and based on the *zigzag* relation defined on the set of local checkpoints. The additional interest of the formulation based on checkpoint intervals lies in the framework it offers to obtain a distributed adaptive checkpointing algorithm.

The second part of the paper has been devoted to the description and to the proof of the adaptive algorithm. It ensures all local checkpoints will participate in some consistent global checkpoints. This algorithm, first, supposes local checkpoints may be taken arbitrarily following some uncoordinated algorithm, then it forces processes to take as few as possible additional local checkpoints in order that all previously taken local checkpoints be useful. This algorithm adds no synchronization and no control messages to the computation and uses only the piggybacking of control information on application messages. This control information consists of a vector of timestamps, a boolean array and a boolean vector. Moreover each time a local checkpoint is taken (either arbitrarily or forced by the algorithm) it is, on-the-fly and without delay, associated with a vector of local checkpoint numbers that define a consistent global checkpoint to which it belongs. Such an adaptive algorithm can be used in many areas such as fault-tolerance, distributed debugging, properties detection, distributed computing etc., where global checkpointing is of primary importance

References

[1] Ö. Babaoğlu, K. Marzullo, Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanism, *Distributed Systems: second edition (Chapter 4)*, Edited by S. Mullender, Addison-Wesley, 1993, pp.55-96.

[2] K.M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems, *ACM Transactions on Computer Systems*, Vol. 3, No. 1, 1985, pp.63-75.

[3] A.P. Goldberg, A. Gopal, A. Lowry, R. Strom, Restoring consistent global states of distributed computations, *Proc. ACM-ONR workshop on Parallel and Dist. Debugging*, Santa-Cruz CA, 1991, Reprinted in *Sigplan Notices*, Vol. 26,12, 1991, pp.144-154.

[4] J.M. Helary, C. Jard, N. Plouzeau, M. Raynal, Detection of stable properties in distributed applications, *Proc. 6th ACM Symposium on Principles of Distributed Computing*, 1987, pp.125-136.

[5] J.M. Helary, M. Raynal, Towards the construction of distributed detection programs with an application to distributed termination, *Distributed Computing*, Vol. 7, 1994, pp.137-147.

[6] R. Koo, S. Toueg, Checkpointing and rollback- recovery for distributed systems, *IEEE Transactions on Software Engineering*, Vol. 13, No. 1, 1987, pp. 23-31.

[7] D.R. Jefferson, Virtual time, *ACM Transactions on Programming Languages and Systems*, Vol. 7,3, 1985, pp.404-425.

[8] D.B. Johnson, W. Zwaenepoel, Recovery in distributed systems using optimistic message logging and checkpointing, *Journal of Algorithms*, Vol. 11, No. 3, 1990, pp. 462-491.

[9] 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.

[10] T. Leblanc, J. M. Mellor-Crummey, Debugging parallel programs with instant replays, *IEEE Transactions on Computers,* Vol. C33-4, 1987, pp.471-482.

[11] R.H.B. Netzer, J. Xu, Adaptive message logging for incremental replay of message passing systems, *Proc. Supercomputing '93*, Portland OR, November 1993, pp.840-849.

[12] R.H.B. Netzer, J. Xu, Necessary and sufficient conditions for consistent global snapshots, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6,2, 1995, pp.165-169.

[13] B. Randell, System structure for software fault tolerance, *IEEE Transactions on Software Engineering*, Vol. SE1-2, 1975, pp.220-232.

[14] D.L. Russell, State Restoration in Systems of Communicating Processes, *IEEE Transactions on Software Engineering*, Vol. SE-6, 1980, pp.183-194.

[15] R.D. Schlichting, F.B., Schneider, Fail-stop processors: an approach to designing fault-tolerant computing systems, *ACM Transactions on Computer Systems*, Vol. 1, 1983, no. 3, pp. 222-238.

[16] R.E. Strom, S. Yemini, Optimistic recovery in distributed systems, *ACM Transactions on Computer Systems* Vol. 3,2, 1985, pp.204-226.

[17] Y.M. Wang, A. Lowry, W.K. Fuchs, Consistent global checkpoints based on direct dependency tracking, *Information Processing Letters*, Vol. 50, 1994, pp.223-230.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399