

# On Modeling Consistent Checkpoints and the Domino Effect in Distributed Systems

Roberto Baldoni, Jean-Michel H elary, Achour Mostefaoui, Michel Raynal

► **To cite this version:**

Roberto Baldoni, Jean-Michel H elary, Achour Mostefaoui, Michel Raynal. On Modeling Consistent Checkpoints and the Domino Effect in Distributed Systems. [Research Report] RR-2569, INRIA. 1995. inria-00074112

**HAL Id: inria-00074112**

**<https://hal.inria.fr/inria-00074112>**

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 destin ee au d ep ot et  a la diffusion de documents scientifiques de niveau recherche, publi es ou non,  emanant des  tablissements d'enseignement et de recherche franais ou  trangers, des laboratoires publics ou priv es.

***On Modeling Consistent Checkpoints and the  
Domino Effect in Distributed Systems***

Roberto Baldoni, Jean Michel Helary  
Achour Mostefaoui , Michel Raynal

**N° 2569**

Juin 1995

PROGRAMME 1



*R*apport  
de recherche





## On Modeling Consistent Checkpoints and the Domino Effect in 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° 2569 — Juin 1995 — 22 pages

**Abstract:** Backward error recovery is one of the most used schemes to ensure fault-tolerance in distributed systems. It consists, upon the occurrence of a failure, in restoring a distributed computation to an error-free global state from which it can be resumed to produce a correct behavior. Checkpointing is one of the techniques to pursue the backward error recovery. In this paper, we present a general framework that takes a semantic including missing and orphan messages into account. Notions of missings and orphans are revisited by considering additional underlying mechanism available on channels and semantics of messages. This framework allows, first, to state and prove a theorem to determine if an arbitrary set of checkpoints is consistent and, second, to define formally the domino effect. Further, we show how previously published uncoordinated checkpointing algorithms can be described in our context and some example of uncoordinated checkpointing algorithms that ensure domino-free rollback recovery are also given.

**Key-words:** Consistent global checkpoints, Lamport's happened-before relation, orphan and missing messages, domino effect, uncoordinated checkpointing algorithms, fault-tolerance, distributed systems.

(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

## Points de reprise et effet domino dans les systèmes répartis

**Résumé :** Ce rapport s'intéresse à la définition de points de reprise dans les systèmes répartis. Un cadre général, intégrant les messages manquants et les messages orphelins, est décrit et une condition nécessaire et suffisante, indiquant si un ensemble de points de reprise locaux est cohérent, est donné. Par ailleurs, une définition formelle de l'effet domino est proposée et des exemples d'algorithmes répartis de détermination de points de reprise cohérents sont présentés.

**Mots-clé :** Points de reprise cohérents, précédence causale de Lamport, messages manquants et messages orphelins, effet domino, algorithme non coordonné, tolérance aux défaillances, systèmes distribués.

## 1 Introduction

*Error recovery* in computer systems is a task that involves restoring an error-free state from an erroneous one. Error recovery schemes are usually classified into *forward* and *backward* error recovery [9]. *Forward error recovery* is based on attempting to make further use of the state which has just been found to be in error. This is possible when the nature of the error and consequences of faults can be completely assessed. Then one can remove those errors and enable the system to move forward. Usually forward error recovery strategies are based on the use of error correcting techniques and codes, or error compensation providing supplementary information. *Backward error recovery* involves, first, backing up one or more of the processes of the system to a previous state, which is hoped to be error-free, before attempting to execute further operations. In this scheme it is not necessary to foresee the nature and consequences of all the faults, and to remove all the errors. Thus, it makes possible to recover from an arbitrary fault simply by restoring the system to an appropriate previous state. These features enable backward recovery to be considered as a general recovery mechanism to any type of fault. However, backward recovery needs *recovery points*, i.e., effective means by which a state of a process can be saved and later restored. For obtaining such recovery points one of the basic techniques is *checkpointing*. It consists in determining, during a computation, a set of local checkpoints (i.e., a set of local states saved in stable storage), from which, upon a failure occurrence, a rolled back computation can be resumed. All distinct pairs of such local checkpoints must be mutually consistent in order that the resumption be feasible. Informally, mutual consistency among local checkpoints ensures that the resumed computation will be semantically correct. Formalization of the notion of consistent global checkpoints can be found in [1, 3, 7].

Many algorithms have been proposed to compute consistent checkpoints. They can be classified into two main families according to the coordination employed to take checkpoints. In the *coordinated approach* [4] processes coordinate explicitly, by means of control messages, the determination of their local checkpoints in order to be mutually consistent. Algorithms that adopt this approach can be seen actually as improvements or adaptations of snapshot algorithms [1] to the checkpointing problem. For each process the last checkpoint taken is always the local state in which the process must be restored to if a failure causes its rollback.

The basic characteristic of the *uncoordinated approach* [10, 12, 14] is that processes take checkpoints more or less independently. However, this approach can be divided into two classes according to the possible occurrence of the domino effect [8] during rollback recovery. The domino effect appears when a subset of processes, which have to be resumed after a failure, rollback unboundedly while determining a set of mutually consistent checkpoints. The first class assume pessimistically that the domino effect could occur and then taking enough checkpoints or logging enough messages is necessary in order not to be subjected to the domino effect [3, 12]. Generally, the more checkpoints and messages are stored in stable storage the less rollback is necessary to determine a set of mutually consistent checkpoints. The second family assume optimistically that the probability of the combined event "failure of a process" and "domino effect" is very close to zero. In these algorithms, processes log only few messages [14]. If a failure occurs, the rollback recovery procedure has to determine from

the checkpoints previously taken, a set of mutually consistent local states for the processes which have to be restarted, but this determination is not guaranteed *a priori* due to the risk of domino effect.

From a performance point of view, these two approaches show an evident tradeoff between the overhead during a distributed computation and the overhead during a recovery. Coordinated checkpointing add to distributed computation checkpointings, message loggings and control messages, thus performances of failure-free computations are penalized in proportion to their size. On the other hand, upon the occurrence of a failure, each process knows the local state in which it must rollback. Taking checkpoints in an uncoordinated way imposes very few overhead during a failure-free distributed computation (e.g. checkpoints may be taken when processes are idle). Upon the occurrence of a failure, the rollback is not *a priori* guaranteed to succeed (i.e., some processes can be rolled back unboundedly) when considering "optimistic" algorithms. In "pessimistic" algorithms there could be a bounded rollback of some processes. However, the rollback is costly in terms of control messages and this cost can be proportional to the size of the distributed computation. For example, if we consider large-scale distributed systems, from previous considerations the coordinated approach is not practicable. At the same time, if the number of processes increases, the probability of the event "failure of a process" and "domino effect" could be no longer negligible. Thus, the uncoordinated approach ensuring domino-freeness seems to be the more appropriate one to cope with large-scale distributed systems.

This paper intends to provide a general framework for intrinsic analysis of backward recovery when considering checkpoints as recovery points. First, we focus on the concept of consistent checkpoints and address very carefully their consistency issues, which are central in backward recovery techniques. At this end, it is shown that, when a *recovery semantic* is associated with messages, the set of local checkpoints from which a resumed computation can be consistently restarted is enlarged. This leads to revisit the notions of *orphan* and *missing* messages by considering the semantic of messages (defined at the application level) and additional underlying mechanisms available on channels (implemented at the system level). Second, we prove a theorem (actually a necessary and sufficient condition) to determine if an arbitrary set of checkpoints is consistent; this condition is very general as it allows to accept or to deny orphan or missing messages in a consistent set of local checkpoints. Further, we give two interesting corollaries on useful checkpoints: one will provide the basic tool for defining formally the domino effect and the other one will give a simple method to obtain domino-free rollback recovery checkpointing algorithms. This framework plays a key role in designing uncoordinated domino-free checkpointing algorithms that takes as less checkpoints as possible and in defining efficient recovery strategies in the presence of bounded domino effect.

The paper is structured into five main sections. Section 2 presents the model of asynchronous distributed executions and introduces the concept of checkpoint interval. Section 3 investigates thoroughly problems related to the presence of orphan and missing messages in a resumed computation, showing their influence on the definition of global checkpoint consistency. In Section 4, a theorem to determine if an arbitrary set of checkpoints is consis-

tent and two corollaries on useful local checkpoints are introduced. Section 5 points out the modeling of the domino effect. Finally, in Section 6, first the well-known Russell's algorithm [10] is described in our context and then some examples of uncoordinated checkpointing algorithms that ensure domino-free recovery are shown.

## 2 System Model

### 2.1 Distributed Computations

A distributed computation consists of a finite set  $P$  of  $n$  processes  $\{P_1, P_2, \dots, P_n\}$  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 [11].

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  $P_j$  actually 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 channel and delivered to  $P_i$ . Executions of send and delivery statements are modeled by send 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  $h_i = e_i^0 e_i^1 \dots e_i^s, \dots$ , 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  $e_i^s$ ;  $h_i^s = e_i^0 e_i^1 \dots e_i^s$  is a prefix of  $h_i$ . Events local to a process are totally ordered. However, the set of all the events of the computation is only partially ordered. Let " $\rightarrow$ " denote the *causal precedence* (*happened-before*) partial order defined as follows ([5]):

**Definition 2.1**  $e_i^s \rightarrow e_j^t$  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 of  $\widehat{H}$ ; the *causal past* of  $e$  is the subset of  $\widehat{H}$  including all events  $e'$  such that  $e' \rightarrow e$ .

Moreover, we consider that processes execute programs which can have non-deterministic statements (i.e., processes' histories cannot be entirely predicted by programs these processes execute). Let us note that this process's behavior includes, but is not limited to, the *piecewise deterministic* one presented in [13] where the only cause of non-determinism is due to the unpredictability of message transmission delays.



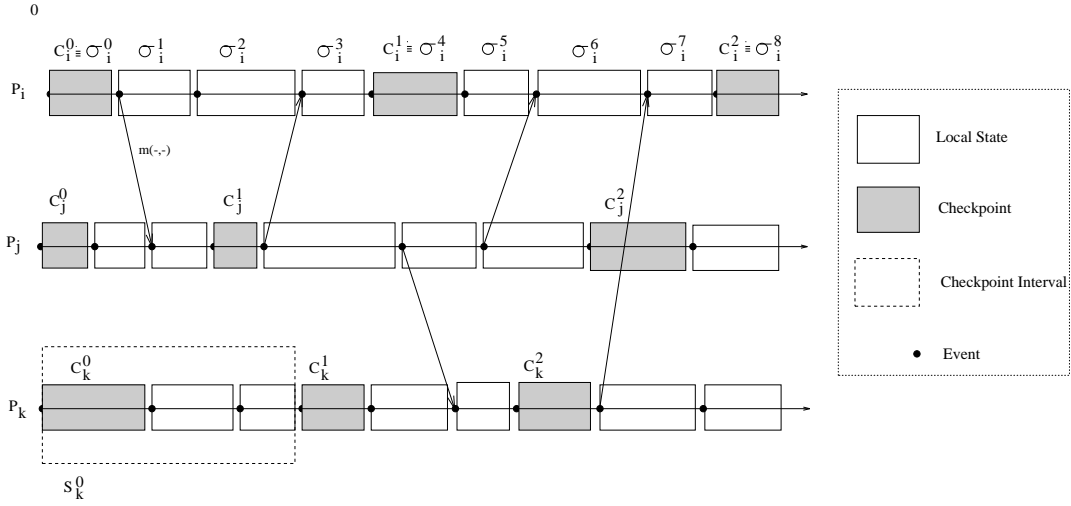


Figure 1: Example of a distributed computation.

## 2.2 Checkpoints and Intervals

### 2.2.1 Checkpoints

Let  $\sigma_i^0$  be the initial state of process  $P_i$ . The event  $e_i^s$  moves  $P_i$  from the local state  $\sigma_i^{s-1}$  to the local state  $\sigma_i^s$ . The local state  $\sigma_i^s$  corresponds to the local history  $h_i^s$ ; by definition  $e_i^x$  belongs to  $\sigma_i^s$  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 *global state* is a set of local states one for each process.

A *local checkpoint*  $C$  is a local state of a process<sup>1</sup> and the set of all local checkpoints is a subset of all local states. Whether a local state is or not a local checkpoint does not depend on the computation itself; checkpoints are determined by an underlying protocol which is superimposed on the computation [2]. In the following we consider a distributed computation  $\widehat{H}$  and a set of local checkpoints  $\mathcal{C}_{\widehat{H}}$  defined on  $\widehat{H}$ .  $C_i^s$  represents the  $s$ -th checkpoint taken by process  $P_i$  and corresponds to some local state  $\sigma_i^x$  with  $s \leq x$ . 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  $\sigma_i^0$ , when it starts.

<sup>1</sup>At the operational level a local checkpoint is a local state that has been stored in stable storage; we say "a process has taken a local checkpoint".

### 2.2.2 Checkpoint Intervals

Using an abstraction level defined by local checkpoints, any two successive local checkpoints  $C_i^s$  and  $C_i^{s+1}$  of process  $P_i$  define an interval. We call *checkpoint interval*  $S_i^s$  the set of events produced by process  $P_i$  between  $C_i^s$  and  $C_i^{s+1}$  (including the event that produced the local state corresponding to  $C_i^s$ ). As an example, Figure 1 depicts the interval  $S_k^0$ .

## 3 Backward Recovery and Global Checkpoints

### 3.1 Backward recovery

Consider distributed computations of a given distributed program, in which fail-stop events can occur. Such an event is a consequence of a system crash implied by an error in a fail-stop processor on which a process runs. Recall that, when an error occurs in a fail-stop processor, it is halted and the content of its volatile storage is lost. To tolerate such faults, *backward error recovery* can be applied. In the case of checkpointing, this technique refers to restoring processes of a distributed computation to previous checkpointed local states, saved on stable storage, from which this computation can be resumed leading to a consistent global state of the restarted computation, not necessarily equivalent to the failed one (e.g. if the program has non-deterministic statements).

### 3.2 The concept of global checkpoints

A *global checkpoint* is defined as a set of local checkpoints, one for each process. Let us note that a global checkpoint constitutes a global state of the computation. In general, different global checkpoints are available. In the context of recovery, we are only interested in *consistent global checkpoints*, whose precise definition needs to introduce the notion of resumed computation and the concept of orphan and missing messages.

### 3.3 Faulty and Resumed computations

In order to address more precisely the issue of global checkpoint consistency, let us introduce some terms. The computation in which a fail-stop event occurs will be called the *faulty computation*; this computation stops in a global state. The backward recovery technique implies that processes are rolled back to local checkpoints forming a consistent global checkpoint. The computation including all events produced till this global checkpoint plus events produced after resumption from this global checkpoint constitute the *resumed computation*. As an example Figure 2 shows a distributed computation, formed by two processes, that stops in the global state  $\{\sigma_i^w, \sigma_j^z\}$  and rolls back to the consistent global checkpoint  $\{C_i^s, C_j^t\}$  from which the resumed computation can restart.

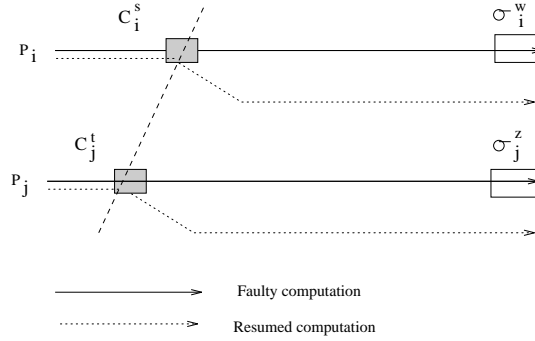


Figure 2: Faulty and Resumed Computation.

### 3.4 Orphan and Missing Messages

#### 3.4.1 Orphan messages

Upon the occurrence of a failure, let us suppose  $P_i$  and  $P_j$  be restarted from checkpoints  $C_i^s$  and  $C_j^t$  respectively.

**Definition 3.1** A message  $m$  sent by  $P_i$  to  $P_j$  is called orphan with respect to the ordered pair of local checkpoints  $(C_i^s, C_j^t)$  if its delivery event belongs to  $C_j^t$  while its sending event does not belong to  $C_i^s$ .

An example of orphan message is shown in Figure 3.a. when considering  $send(m) \notin C_i^s$  and  $delivery(m) \in C_j^t$ . When the computation restarts from  $C_i^s$  and  $C_j^t$  according to the non-deterministic behavior of  $P_i$  two cases are possible:

- i) *the event  $send(m)$  is re-executed in the resumed computation*: the message  $m$  is delivered two times to  $P_j$  affecting then  $P_j$ 's local state twice. However, if the multiple delivery of  $m$ , at the application level, does not affect  $P_j$ 's correctness [6] or if, at the system level, an additional underlying mechanism discarding duplicated messages is available [3] then the message  $m$ , orphan with respect to  $(C_i^s, C_j^t)$ , will no longer remain orphan with respect to ordered pairs of checkpoints  $(C_i^x, C_j^y)$  with  $x > s$  and  $y \geq t$  as in the resumed computation  $m$  has been sent once and received once;
- ii) *the event  $send(m)$  is not re-executed in the resumed computation*: then the message  $m$  remains definitely orphan.

In some cases the application can tolerate orphan messages [6]. Even though the application does not tolerate orphans, as seen above, when the system can recover from such messages they can be momentarily tolerated since they will no longer remain orphan in the future.

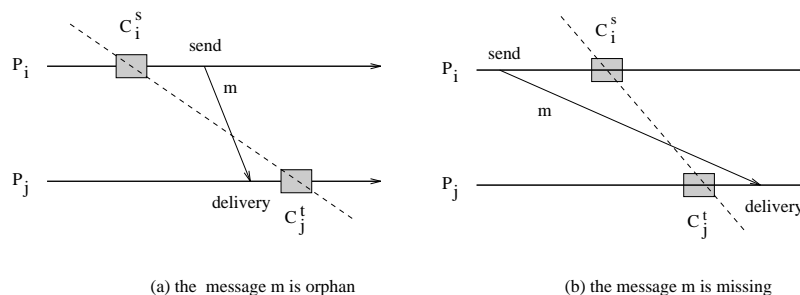


Figure 3: Orphan and Missing messages.

### 3.4.2 Missing messages

Upon the occurrence of a failure, let us suppose  $P_i$  and  $P_j$  be restarted from checkpoints  $C_i^s$  and  $C_j^t$  respectively.

**Definition 3.2** A message  $m$  sent by  $P_i$  to  $P_j$  is called missing with respect to the ordered pair of local checkpoints  $(C_i^s, C_j^t)$  if its sending event belongs to  $C_i^s$  and its delivery event does not belong to  $C_j^t$ .

An example of missing message is shown in Figure 3.b. where  $send(m) \in C_i^s$  and  $delivery(m) \notin C_j^t$ . When the computation restarts from  $C_i^s$  and  $C_j^t$  according to the non-deterministic behavior of  $P_i$  two cases are possible:

- iii) *the event  $delivery(m)$  is re-executed in the resumed computation*: in order that this re-execution be feasible, the message  $m$  (as its sending is not re-executed) has to be recorded by the system level or logged somewhere on non-corrupted storage. If this recording has not been done the message  $m$  remains missing;
- iv) *the event  $delivery(m)$  is not re-executed in the resumed computation*: then  $m$  remains missing forever.

As previously, in some cases the application can tolerate missing messages. Even though the application does not tolerate missings, when the system can recover from such messages (by considering states of channels [1, 3, 4, 12]) missings can be momentarily tolerated since they will no longer remain missing in the future.

## 3.5 Analyzing Consistency of Global Checkpoints

An ordered pair  $(C_i^s, C_j^t)$  is consistent if all messages sent by  $P_i$  to  $P_j$  till  $C_i^s$  have been delivered to  $P_j$  before  $C_j^t$ . The previous discussion (cases i,ii,iii,iv) shows that, in some

circumstances, such a pair  $(C_i^s, C_j^t)$  could be considered consistent even though it includes sending events of messages and not the corresponding delivery events or vice versa. To allow such possibilities, messages are tagged and the formal definition of consistency takes such tags into account. Next Subsections introduce tagging of messages and consistency of global checkpoints.

### 3.5.1 Tagging of messages

A *tag* is associated with each message; it indicates if the message can be orphan and/or missing with respect to pairs of local checkpoints. Each message is tagged with a pair of tags  $cb_o, cb_m$ , leading to four different status: *orphan or missing*, *orphan but not missing*, *missing but not orphan*, *neither orphan nor missing*. Thus, four basic types of message deliveries are possible:

- *Exactly once* message delivery (messages are tagged  $(-, -)$ , i.e., messages cannot be either orphan or missing),
- *At least once* message delivery (messages are tagged  $(cb_o, -)$ , i.e., messages can be orphan but not missing),
- *At most once* message delivery (messages are tagged  $(-, cb_m)$ , i.e., messages can be missing but not orphan),
- *No constraint* on message delivery (messages are tagged  $(cb_o, cb_m)$ , i.e., messages can be either missing or orphan).

Tags can be set either by the application itself, by a compiler or by the underlying system:

**static tagging** : a message, that for the application's semantic can be orphan, will be tagged  $cb_o$  by the application or by the compiler. Consequently, if this message is orphan with respect to a pair of checkpoints, it may remain orphan in the future.

**dynamic tagging** : if the system maintains information about the behavior of the application (e.g. deterministic, piecewise deterministic or non-deterministic) and it is able to discard duplicates, then the system can tag messages  $cb_o$  as it ensures they will remain no longer orphans in the future.

The same discussion holds for tagging messages as missing ( $cb_m$ ). In the following we assume that a message tagged  $(-, *)$  is a message that cannot be orphan but it may be missing and that a message tagged  $(*, -)$  is a message that cannot be missing but it may be orphan.

### 3.5.2 Consistency of a global checkpoint

The definition of the consistency of global checkpoint lies on tags associated with each message. These tags allow to introduce a reflexive relation of exclusion, denoted  $\overset{ex}{\rightarrow}$ , on local

checkpoints. Let us consider  $C_i^s$ ; informally, the relation  $C_i^s \xrightarrow{ex} C_j^t$  means that no consistent global checkpoint can include a pair of local checkpoints  $C_i^x$  and  $C_j^y$  with  $x \leq s$  and  $y \geq t$  respectively. More formally:

**Definition 3.3** *Let  $(C_i^s, C_j^t)$  be an ordered pair of local checkpoints.  $C_i^s \xrightarrow{ex} C_j^t$  if and only if there exists a message  $m$  such that:*

- $m$  is not tagged  $cb\_o$  and the sending of  $m$  belongs to  $C_i^x$  ( $x > s$ ) and the delivery of  $m$  belongs to  $C_j^y$  ( $y \leq t$ ), or
- $m$  is not tagged  $cb\_m$  and the sending of  $m$  belongs to  $C_j^t$  ( $y \leq t$ ) and the delivery of  $m$  belongs to  $C_i^x$  ( $x > s$ ).

The relation of exclusion shows that a local checkpoint,  $C_i^s$ , excludes another one,  $C_j^t$ , if there exists either a missing or an orphan message which cannot be accepted by the resumed computation restarted from  $C_i^s$  and  $C_j^t$ . As an example, Figure 4 shows that the existence either of the message  $m'$ , orphan and not tagged ( $cb\_o, -$ ), or of the message  $m''$ , missing and not tagged ( $-, cb\_m$ ), implies that  $C_i^s$  excludes  $C_j^t$ .

Moreover, from this definition, it is simple to see that  $C_i^s$  excludes  $C_j^t$  implies that all checkpoints that are on the past of  $C_i^s$  exclude, transitively, the ones on the future of  $C_j^t$  as depicted in Figure 4. More formally,

$$C_i^s \xrightarrow{ex} C_j^t \wedge x \leq s \wedge y \geq t \Rightarrow C_i^x \xrightarrow{ex} C_j^y$$

In that sense, it is worth noting that the relation of exclusion is not symmetric. A counter example is shown in Figure 4 where  $C_i^s \xrightarrow{ex} C_j^t$  but  $\neg(C_j^t \xrightarrow{ex} C_i^s)$ : in fact, the local checkpoint  $C_j^{t-1}$  belonging to the past of  $C_j^t$  does not exclude the local checkpoint  $C_i^{s+1}$  belonging to the future of  $C_i^s$ , since, with respect to the resumed computation restarted from  $C_i^{s+1}$  and  $C_j^{t-1}$ ,  $m'$  is missing and tagged ( $-, cb\_m$ ) and  $m''$  is orphan and tagged ( $cb\_o, -$ ). However,  $C_i^s$  and  $C_j^t$  cannot clearly belong to the same consistent global checkpoint. We are then able to give the definition of consistency for an ordered pair of local checkpoints and of consistency of a global checkpoint:

**Definition 3.4** *An ordered pair of local checkpoints  $(C_i^s, C_j^t)$  is consistent if  $\neg(C_i^s \xrightarrow{ex} C_j^t)$ .*

**Definition 3.5** *A global checkpoint,  $\{C_1^{x_1}, C_2^{x_2}, \dots, C_n^{x_n}\}$  is consistent with respect to  $\xrightarrow{ex}$  if and only if, for every  $(i, j)$  such that  $1 \leq i \leq n, 1 \leq j \leq n$  and  $i \neq j$ , the ordered checkpoint pair  $(C_i^{x_i}, C_j^{x_j})$  is consistent.*

## 4 Relation of Precedence on Checkpoint Intervals

This Section defines a relation of precedence, denoted  $\prec$ , on checkpoint intervals. This relation will allow to state a necessary and sufficient condition to determine whether an arbitrary

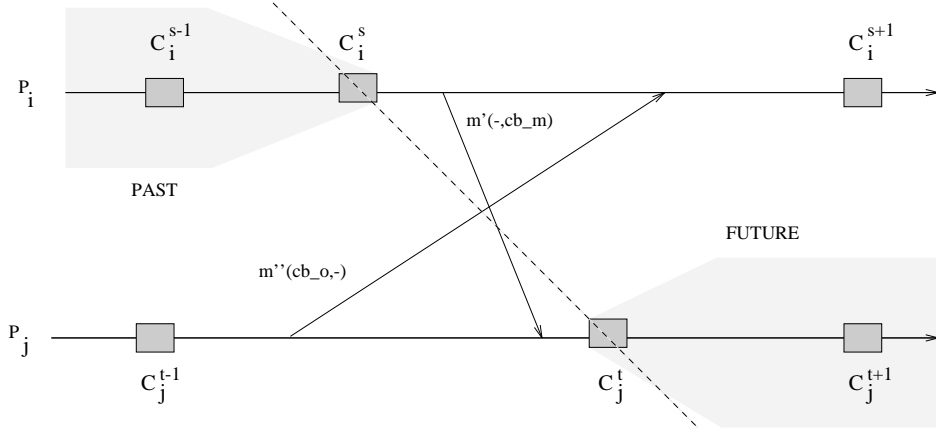


Figure 4: Example of the relation of exclusion.

set of local checkpoints can belong to some consistent global checkpoint. Further, we will present an interesting corollary stating when a local checkpoint can belong to some consistent global checkpoint and, finally, we show that our framework includes the *zigzag* relation introduced by Netzer and Xu [7].

#### 4.1 Precedence Relation

**Definition 4.1**  $S_i^s$  precedes  $S_j^t$  (denoted  $S_i^s \prec S_j^t$ ) if and only if:

1.  $i = j$  and  $t = s$ , or
2.  $i = j$  and  $t = s + 1$ , or
3.  $C_i^s \xrightarrow{ex} C_j^{t+1}$ , or
4.  $\exists S_k^u : S_i^s \prec S_k^u \wedge S_k^u \prec S_j^t$ .

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 5 shows the graph of the relation  $\prec$  of the computation of Figure 1 when considering all the messages are tagged  $(-, -)$  and where, for charity's sake, we do not consider transitive precedences due to point 4 of Definition 4.1. In this graph, for example, the edge  $S_i^0 \prec S_j^0$  is due to message  $m$  of Figure 1 sent in  $S_i^0$  and delivered in  $S_j^0$  ( $C_i^0 \xrightarrow{ex} C_j^1$  and  $m$  cannot be orphan). The message  $m$  implies also the opposite edge  $S_j^0 \prec S_i^0$  ( $C_j^0 \xrightarrow{ex} C_i^1$  and  $m$  cannot be missing).

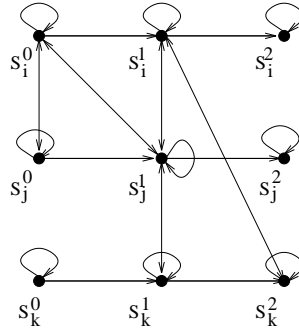


Figure 5: Relation of precedence on checkpoint intervals.

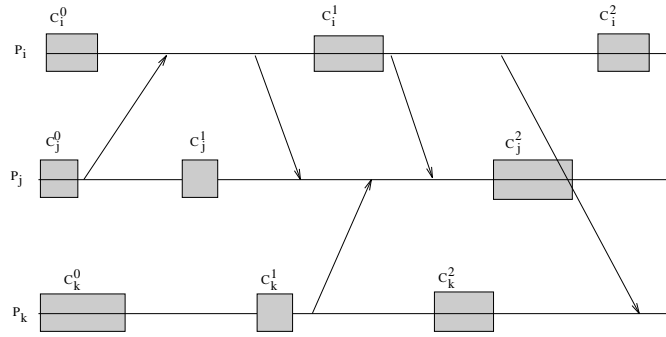


Figure 6: A distributed computation equivalent to the one of Figure 1

Before introducing the consistency Theorem, let us note that, since the presence of either an orphan or a missing message is equivalent to state the inconsistency of a checkpoint pair (Section 3.5.2), the graph of the precedence relation  $\prec$  actually represents a broad class of equivalent distributed computations. For example the distributed computation of Figure 6, when considering all the messages are tagged  $(-, -)$ , is equivalent (with respect to  $\prec$ ) to the one depicted in Figure 1 since they have the same precedence relation graph shown in Figure 5.

### 4.2 Consistency Theorem

**Theorem 4.1** *Let  $\mathcal{I} \subseteq \{1, \dots, n\}$  and  $\mathcal{C} = \{C_i^{x_i}\}_{i \in \mathcal{I}}$  be a set checkpoints. Then  $\mathcal{C}$  is a subset of a consistent global checkpoint if and only if:*

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

**Proof**



**Sufficiency.** We prove that if  $(CT)$  is satisfied then  $\mathcal{C}$  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). 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  $\exists k \in \mathcal{I} :: S_i^{x_i-1} \prec S_k^{x_k-1}$ .

We show that  $\{C_1^{x_1}, C_2^{x_2}, \dots, C_n^{x_n}\}$  is consistent with respect to  $\xrightarrow{\epsilon x}$ . Assume the contrary. There exists  $i$  and  $j$  such that  $C_i^{x_i} \xrightarrow{\epsilon x} C_j^{x_j}$  and thus from point 3 of Definition 4.1 we have:

$$S_i^{x_i} \prec S_j^{x_j-1} \quad (1)$$

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  $(CT)$ ;
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  $\{C_1^{x_1}, C_2^{x_2}, \dots, C_n^{x_n}\}$  including  $\mathcal{C}$  then property  $(CT)$  holds for any  $\mathcal{I} \subseteq \{1, \dots, n\}$ . Assume the contrary. There exists  $i \in \mathcal{I}$  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 sequence of integer  $i = i_1, i_2, \dots, i_k = j$  and a sequence of checkpoint numbers  $y_1 = x_i, y_2, \dots, y_k = x_j$  such that

$$C_i^{x_i} \xrightarrow{\epsilon x} C_{i_2}^{y_2} \text{ and } C_{i_2}^{y_2} \xrightarrow{\epsilon x} C_{i_3}^{y_3} \text{ and } \dots \text{ and } C_{i_{k-1}}^{y_{k-1}} \xrightarrow{\epsilon x} C_j^{x_j}$$

We show by induction on  $k$  that  $\forall t \geq x_j$ ,  $C_i^{x_i}$  and  $C_j^t$  cannot belong to the same global checkpoint<sup>2</sup>.

*Base step.*  $k = 2$ . In this case,  $C_i^{x_i} \xrightarrow{\epsilon x} C_j^{x_j}$  and thus,  $\forall t \geq x_j$ ,  $C_i^{x_i} \xrightarrow{\epsilon x} C_j^t$ . Consequently,  $C_i^{x_i}$  and  $C_j^t$  cannot belong to the same consistent global checkpoint.

<sup>2</sup>In the case  $i = j$  this relation follows from the reflexivity of relation  $\xrightarrow{\epsilon x}$ .

*Induction step.* We suppose the result true for some  $k \geq 2$  and show that it holds for  $k + 1$ . We have:

$$C_i^{x_i} \xrightarrow{ex} C_{i_2}^{y_2} \text{ and } C_{i_2}^{y_2} \xrightarrow{ex} C_{i_3}^{y_3} \text{ and } \dots \text{ and } C_{i_k}^{y_k} \xrightarrow{ex} C_j^{x_j}$$

From the assumption induction, for any  $t \geq y_k$ ,  $C_i^{x_i}$  and  $C_{i_k}^t$  cannot belong to the same consistent checkpoint. Moreover,  $C_{i_k}^{y_k} \xrightarrow{ex} C_j^{x_j}$  implies that, for any  $u \leq y_k$ ,  $C_{i_k}^u$  and  $C_j^{x_j}$  cannot belong to the same consistent checkpoint. Therefore, no checkpoint in process  $P_{i_k}$  can be included with  $C_i^{x_i}$  and  $C_j^{x_j}$  to form a global consistent checkpoint.  $\square$

### 4.3 Useful Local Checkpoints

The previous Theorem has an interesting corollary which allows to decide whether a local checkpoint can be a member of a consistent global checkpoint:

**Corollary 4.2** *A checkpoint  $C_i^s$  can belong to some consistent global checkpoint if and only if:*

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

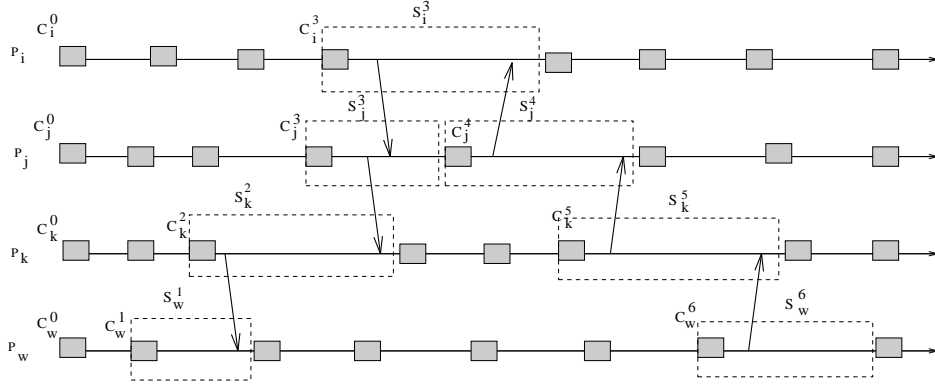
Checkpoints that verify the previous corollary are called *useful*. Non-useful checkpoints cannot belong to any consistent global checkpoint and they are the direct cause of the rollback that may lead, during the recovery, to the domino effect. For example, checkpoints  $C_i^1$ ,  $C_j^1$  and  $C_k^2$  of Figure 1 are non-useful as it results from Figure 5. For useless checkpoints we can state the following corollary that will be used in the following:

**Corollary 4.3**  *$C_i^s$  is a useless checkpoints (i.e.,  $S_i^s \prec S_i^{s-1}$ )  $\Leftrightarrow \exists j : S_i^s \prec S_j^l$  and  $S_j^l \prec S_i^{s-1}$ .*

Let us note, finally, that when we consider the particular case where all the messages are tagged  $(-, cb\_m)$ , results of Theorem 4.1 and Corollary 4.2 simplify and are theoretically equivalent to the ones proposed by Netzer and Xu in [7] using the *zigzag* relation. The additional interest of our framework is twofold: first, the relation given in Corollary 4.2 provides the basic tool for modeling the domino effect as we will describe in the next Section and, second, Corollary 4.3 provides a simple method to obtain domino-free rollback recovery checkpointing algorithms as we will see in Section 6.

## 5 Characterizing the Domino Effect

This Section models the domino effect by using the relation of precedence on checkpoint intervals. The domino effect forces processes to rollback unboundedly. Such a "backward progress" of a process, during recovery, is captured by Corollary 4.2 where for non-useful

Figure 7: A distributed computation with  $\alpha$ -bounded domino effect

checkpoints we have  $S_i^s \prec S_i^{s-1}$ . In other words, the existence of a non-useful checkpoint  $C_i^s$  means that a checkpoint interval  $S_i^s$  precedes another one  $S_i^{s-1}$  that belongs to its past. As an example, the checkpoint  $C_k^2$  of the distributed computation shown in Figure 1 corresponds to the precedence  $S_k^2 \prec S_k^1$ , transitive consequence of  $S_k^2 \prec S_i^1$  and  $S_i^1 \prec S_k^1$  (Figure 5). This suggests to define the notion of the bounded domino effect:

**Definition 5.1** Let  $\widehat{H}$  be a distributed computation,  $\mathcal{C}_{\widehat{H}}$  be a set of checkpoints defined on  $\widehat{H}$  and  $\alpha \geq 0$  be an integer. The domino effect is  $\alpha$ -bounded in  $\mathcal{C}_{\widehat{H}}$  if and only if:

$$\forall P_i \in P : (S_i^s \prec S_i^t \wedge t < s) \Rightarrow s - t \leq \alpha$$

If  $\mathcal{C}_{\widehat{H}}$  is such that the domino effect is  $\alpha$ -bounded, we say that  $\mathcal{C}_{\widehat{H}}$  is  $\alpha$ -bounded. Consider, as an example, the distributed computation and its relation on checkpoint intervals  $\prec$  depicted in Figures 7 and 8 respectively (for clarity's sake, all the messages are tagged  $(-, cb\_m)$ , and precedences due to reflexivity and transitivity are not shown). In this computation, orphan messages and the useless checkpoint  $C_j^4$  ( $S_j^4 \prec S_j^3$ ) create, by transitivity, the relation of precedence  $S_w^6 \prec S_w^1$  which fixes an upper bound for the domino effect in that computation with these checkpoints:  $\mathcal{C}_{\widehat{H}}$  is 5-bounded.

$\mathcal{C}_{\widehat{H}}$  is defined on the fly by a checkpointing algorithm as  $\widehat{H}$  progresses. So neither  $\widehat{H}$  nor  $\mathcal{C}_{\widehat{H}}$  are known in advance. Suppose that, for a given checkpointing algorithm, we are able to produce a non-negative integer  $\alpha$  such that, for any  $\widehat{H}$ , it will produce a set  $\mathcal{C}_{\widehat{H}}$  which is  $\alpha$ -bounded; then, we know that, in case of failure, a recovery will not require processes to rollback unboundedly. This consideration provides the following definition for domino boundedness.

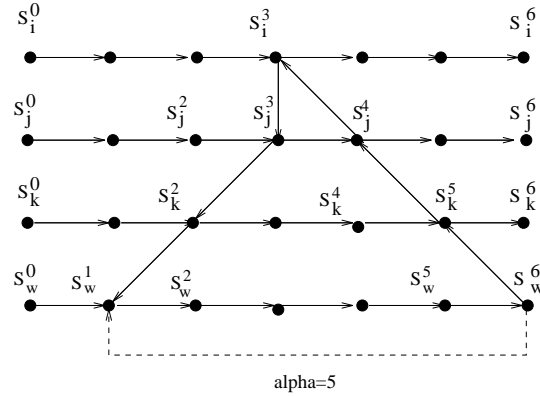


Figure 8: Relation on checkpoint intervals of the computation shown in Figure 7

**Definition 5.2** A checkpointing algorithm ensures domino-bounded rollback recovery if it is possible to define an integer  $\alpha \geq 0$  such that, for any  $\widehat{H}$ , the set  $\mathcal{C}_{\widehat{H}}$  produced by this algorithm is  $\alpha$ -bounded.

A particular interesting case is domino freedom:

**Definition 5.3** A checkpointing algorithm ensures domino-free rollback recovery if for any  $\widehat{H}$ , the set  $\mathcal{C}_{\widehat{H}}$  produced by this algorithm is 0-bounded.

For domino-free rollback recovery algorithms, let us introduce the following simple lemma whose proof follows directly from Corollary 4.2 and from Definitions 5.1 and 5.3:

**Lemma 5.1** A checkpointing algorithm ensures domino-free rollback recovery if and only if it does not produce useless checkpoints.

## 6 Examples of Checkpointing Algorithms

In this Section a sufficient condition to ensure that all local checkpoints be useful is shown and then we propose three uncoordinated and pessimistic checkpointing algorithms that take the extended semantic for messages into account and we show that they ensure domino-free rollback recovery. First, we address the well-known Russell’s algorithm [10] showing how it can be described in our context. Then, a trivial checkpointing algorithm is presented. Finally, we give an improved version of the trivial algorithm which reduces the number of checkpoints taken while ensuring domino-free rollback recovery. All these algorithms are particularly important since they adopt a purely local strategy: each process does not hold

any information about the state of the other processes (i.e., no control information is piggybacked on application messages) and events of each process are sequenced by the same regular expression [10].

## 6.1 A sufficient condition to ensure checkpoint usefulness

Before showing algorithms, let us introduce notations and a property that will be used throughout this Section. Due to communications, four types of attributes can be associated with communication events:

- The attribute *sno* is associated with *send(m)* whose  $tag(m) = (-, *)$ ;
- The attribute *dno* is associated with *delivery(m)* whose  $tag(m) = (-, *)$ ;
- The attribute *dnm* is associated with *delivery(m)* whose  $tag(m) = (*, -)$ ;
- The attribute *snm* is associated with *send(m)* whose  $tag(m) = (*, -)$ ;

Moreover, we assume the attribute "c" corresponds to the "take a checkpoint" operation. Let us now introduce the following property that shows which are the sequences of attributes that occur in a checkpoint interval whenever there exists a useless checkpoint. It is clear that all local checkpoints are useful if no one of these sequences ever appear in any checkpoint interval.

**Property 6.1** *If  $C_i^s$  is a useless checkpoint then there exists a checkpoint interval  $S_j^l$  in which at least one of the following sequence of events occurs:*

*dno sno or dno dnm or snm sno or snm dnm or sno dno or sno snm or dnm snm or dnm dno*

**Proof** if  $C_i^s$  is a useless checkpoint (i.e.,  $S_i^s \prec S_i^{s-1}$ ), then from Corollary 4.3 we have  $\exists j : S_i^s \prec S_j^l$  and  $S_j^l \prec S_i^{s-1}$ .

- As  $S_j^l \prec S_i^{s-1}$  then, from Definitions 4.1 and 3.3, there exists a sequence of messages  $m_1, m_2, \dots, m_q$  with  $q \geq 1$  such that  $S_j^l \prec S_1^{x_1} \prec \dots \prec S_{q-1}^{x_{q-1}} \prec S_i^{s-1}$  and either an event of type *sno* or an event of type *dnm* belongs to  $S_j^l$  (Figure 9.a shows the case with  $q = 1$ ).
- As  $S_i^s \prec S_j^l$  then, from Definitions 4.1 and 3.3, there exists a sequence of messages  $m_1, m_2, \dots, m_q$  with  $q \geq 1$  such that  $S_i^s \prec S_1^{x_1} \prec \dots \prec S_{q-1}^{x_{q-1}} \prec S_j^l$  and either an event of type *snm* or an event of type *dno* belongs to  $S_j^l$  (Figure 9.b shows the case with  $q = 1$ ).

Hence, if there exists a useless checkpoint then one of the following eight sequences of events occurs in a process: *dno sno*<sup>3</sup> or *dno dnm* or *snm sno* or *snm dnm* or *sno dno* or *sno snm* or *dnm snm* or *dnm dno*. Then, the claim follows.  $\square$

<sup>3</sup>Note that this sequence of attributes can physically never occur.

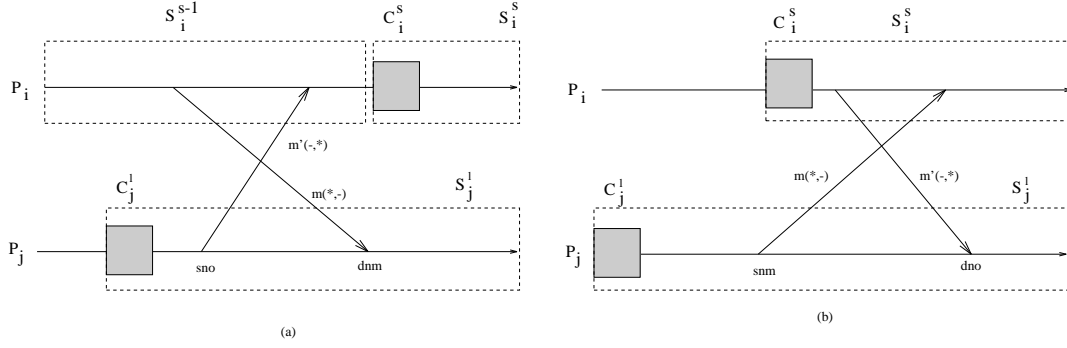


Figure 9: Proof of Properties 6.1

Hence, to guarantee that all checkpoints be useful, each process must prevent the occurrence of sequences of attributes shown in Property 6.1 by taking additional checkpoints. This behavior can be well described by regular expressions defined over the alphabet formed by attributes.

## 6.2 Russell's Algorithm

Russell's checkpointing algorithm [10] is a well-known domino-free algorithm that forces processes to take checkpoints according to this regular expression defined over the alphabet  $\{s, d, c\}$ :  $[c d^* s^*]^*$ , when considering  $s$  and  $d$  be the sending and the delivery event of a message respectively. This expression gives the sequence of events that a process has to produce and it was originally derived considering only the avoidance of orphan messages as the consistency criterion. In our context, this means that all the messages of the distributed computation are tagged  $(-, cb\_m)$  and the previous regular expression becomes the following one defined over the alphabet of attributes  $\{sno, dno, c\}$ :

$$[c dno^* sno^*]^*$$

In [10], it has been proved that the process behavior forced by Russell's regular expression is sufficient to ensure domino-free rollback recovery. However, this can be easily proved using our framework. By considering only the avoidance of orphan messages, from Property 6.1 only two sequence of events must be avoided:  $dno sno$  and  $sno dno$ . The first sequence is impossible to occur and the second is prevented by Russell's regular expression so from Lemma 5.1 Russell's algorithm is domino-free rollback recovery<sup>4</sup>.

<sup>4</sup>This proof of Russell's algorithm shows that it works without assuming FIFO communication between processes as Russell did in [10].

### 6.3 A Checkpointing Algorithm

As we consider messages with arbitrary tags, a trivial solution to the uncoordinated checkpointing problem consists in forcing each process to take atomically a checkpoint after the send and the receive event of each message that cannot be orphan (i.e., messages tagged  $(-, *)$ ) and after each send and delivery event of each message that cannot be missing (i.e., messages tagged  $(*, -)$ ). Such a process behavior can be easily described by the following regular expression defined over the alphabet  $\{sno, snm, dno, dnm, c\}$  giving the sequence of types of attributes that a process has to produce:

$$[c (sno \mid dnm \mid dno \mid snm)]^*$$

The previous regular expression prevents all sequences of Property 6.1 in each process and so from Lemma 5.1 the trivial algorithm ensures *domino-free rollback recovery*.

Now, we show a domino-free rollback recovery algorithm, called  $\mathcal{L}$ , that reduces the number of checkpoints taken compared to the trivial solution. In this solution, each process  $P_i$  works in two basic modes, *mode\_1* and *mode\_2*, and obeys the following rules:

1. If  $P_i$  is in *mode\_1* and sends a message tagged  $(-, *)$  or a message tagged  $(*, -)$  is delivered, then it takes a checkpoint and switches to *mode\_2* before executing this communication event;
2. If  $P_i$  is in *mode\_2* and sends a message tagged  $(*, -)$  or a message tagged  $(-, *)$  is delivered, then it takes a checkpoint and switches to *mode\_1* before executing this communication event.

Such a process behavior is described by the following regular expression defined over the alphabet  $\{sno, snm, dno, dnm, c\}$ :

$$[c ( (snm \mid dno)^* \mid (sno \mid dnm)^* ) ]^*$$

From the regular expression, it is simple to see that  $\mathcal{L}$  includes the trivial one and that the number of checkpoints taken by  $\mathcal{L}$  is less than or equal to the trivial one. It is equal when considering the particular case in which attributes of type *snm* or *dno*, and attributes of type *sno* or *dnm* are alternated in each process.

**Property 6.2** *The regular expression of  $\mathcal{L}$  ensures domino-free rollback recovery.*

**Proof** No one of the sequences of Property 6.1 can ever be produced by the regular expression associated with  $\mathcal{L}$ . Then, from Lemma 5.1 the claim follows.  $\square$

## 7 Conclusion

Checkpointing is one of the techniques to pursue backward error recovery in distributed systems. It consists, upon the occurrence of a failure, in restoring a distributed computation

in a consistent global checkpoint from which it can be restarted to produce a correct behavior. In this paper, we have presented a general framework that lies on a precedence relation on checkpoints interval and that takes a semantic including missing and orphan messages into account.

Notions of missings and orphans have been revisited by considering additional underlying mechanism available on channels and semantics of application messages. We have proved a theorem (actually a necessary and sufficient condition) to determine if an arbitrary set of checkpoints is consistent; this condition is very general as it allows to accept or to deny orphan or missing messages in a consistent set of local checkpoints.

From a theoretical viewpoint this analysis includes the one introduced in [7]: this study, based on the *zigzag* relation defined on the set of local checkpoints, assumes that missings are always accepted and orphans are always denied. Moreover, the additional interest of our framework is twofold: the relation given in Corollary 4.2 have provided the basic tool for modeling the domino effect and, second, Corollary 4.3 have provided a simple method to construct domino-free rollback recovery uncoordinated checkpointing algorithms. Further, we have shown how the well-known Russell's checkpointing algorithm [10] can be described in our context.

As shown in last Section, this framework can play a key role in designing uncoordinated checkpointing algorithms that require to take as less additional checkpoints as possible ensuring domino-free recovery and in designing checkpointing algorithms with bounded rollback recovery.

## Acknowledgements

The Authors are grateful to Jerzy Brzeziński for his support in generalizing the semantic of missing and orphan messages. A special thank goes to Bruno Ciciani for interesting discussions about the checkpointing problem and the application of our framework to Russell's algorithm.

## References

- [1] 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.
- [2] K.M. Chandy, J. Misra, *Parallel program design: a foundation*, Addison Wesley, New York, 1988.
- [3] 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.
- [4] R. Koo, S. Toueg, Checkpointing and rollback- recovery for distributed systems, *IEEE Transactions on Software Engineering*, Vol. 13, No. 1, 1987, pp. 23-31.



- [5] 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.
- [6] H.V. Leong, D. Agrawal, Using message semantics to reduce rollback in optimistic message logging recovery scheme, *Proc. 14-th IEEE International Conference on Distributed Computing Systems*, Poznan, 1994, pp. 227-234.
- [7] R.H.B. Netzer, 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.
- [8] B. Randel, System structure for software fault tolerance, *IEEE Transactions on Software Engineering*, SE-1, No. 2, 1975, pp. 220-232.
- [9] B. Randel, P.A. Lee, P.C., Treleaven, Reliability issues in computing system design, *Computing Surveys*, Vol. 10, No. 2, 1978, pp. 123-165.
- [10] D.L. Russell, State Restoration in Systems of Communicating Processes, *IEEE Transactions on Software Engineering*, Vol. SE-6, 1980, pp.183-194.
- [11] 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.
- [12] R.E. Strom, S. Yemini, Optimistic recovery in distributed systems, *ACM Transactions on Computer Systems* Vol. 3, No. 2, 1985, pp.204-226.
- [13] R.E. Strom, D.F. Bacon, S.A. Yemini, Volatile logging in n-fault-tolerant distributed systems, *Proc. 18-th International Conference of Fault Tolerant Computing Systems*, 1988, pp. 44-49.
- [14] J. Xu, R.H.B. Netzer, Adaptive independent checkpointing for reducing rollback propagation, *Proc. 4-th IEEE Symposium on Parallel and Distributed Processing*, 1993, pp.754-761.



---

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, Irista, 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