Déterminer un état global

Jean-Michel Hélary, Achour Mostefaoui, Michel Raynal

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Déterminer un état global dans un système réparti

J.-M. Hélary, A. Mostefaoui et M. Raynal

N° 2090

Novembre 1993

PROGRAMME 6

Programme 1

ISSN 0249-6399

Déterminer un état global dans un système réparti

J.-M. Hélary, A. Mostefaoui, M. Raynal

Programme 1- Architectures parallèles, bases de données.
réseaux et systèmes distribués

Projet Algorithmes Distribués et Protocoles

Rapport de recherche n˚2090 - Novembre 1993 - 21 pages

Résumé : Le calcul d'un état global d'un système réparti est l'un des paradigmes des problèmes de contrôle réparti. Après avoir exposé le problème, cet article en analyse et présente plusieurs solutions. Celles-ci se distinguent les unes des autres par les hypothèses qu'elles font sur les canaux de communication et les messages de contrôle utilisés. En plus de son caractère "synthétique" sur le calcul d'un état global réparti, cet article illustre de nombreuses hypothèses (et difficultés) propres au contexte réparti.

Mots-clé : cohérence, état global, message de contrôle, ordre causal, système réparti asynchrone.

Ce travail a bénéficié du soutien du CNET au titre du marche 92 1B 178 (Conception formelle de systèmes de coopération haut débit multimédia, projet CESAME).

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : (33) 99 84 71 00 - Télécopie : (33) 99 38 38 32

Determining a Global State in a Distributed System

Abstract: Recording a global snapshot of a distributed system is one of the paradigms of distributed control problems. This paper presents and analyses several solutions. They differ by the assumptions made on the communication channels and the use of control messages. Moreover, this paper throws light on assumptions and difficulties inherent to a distributed environment.

Key-words: causal order, consistency, global state, control message, asynchronous distributed system.

1 Introduction

Parmi les services attendus de la part d'un système, l'observation des programmes en exécution est importante à plusieurs titres: elle peut permettre de détecter des propriétés caractérisant l'exécution (passage dans un état particulier, vérification d'assertions, terminaison, interblocage, etc), d'effectuer des mesures de performances, de capter des états pouvant servir de points de reprise en cas de défaillances ultérieures de l'exécution ou dans le cadre d'un metteur au point, etc. Dans un contexte de répartition, l'observation des programmes en exécution présente des difficultés qui rendent le problème non trivial. En effet, un programme réparti est constitué d'un ensemble de processus s'exécutant en parallèle et communiquant seulement par envoi de messages sur des canaux les interconnectant. En particulier, il n'y a pas d'horloge commune aux différents processus, et de plus, les temps de transfert des messages ne sont pas bornés (dans un contexte de communication fiable, ils sont toutefois finis). Dans ces conditions, il est impossible d'effectuer une observation "simultanée" de l'état des différents processus et canaux. La réalisation d'une observation cohérente qui reflète un état global constitué des états locaux des différentes parties du système, pris à des instants physiques différents mais de manière à rendre une information utile sur l'état du système dans son intégralité, constitue donc un problème désormais classique, connu sous le nom de détection d'un état global réparti cohérent.

Chandy et Lamport illustrent ainsi la détection d'un état global d'un système [7]. L'algorithme de détection d'un état global joue le rôle d'un groupe de photographes qui observent une scène dynamique, par exemple un vol d'oiseaux migrateurs. La scène est si vaste qu'elle ne peut être prise par un seul photographe. Différents photographes doivent donc se charger de prendre chacun une partie de la scène tout en sachant que les photographies ne peuvent être prises au même instant et en exigeant que les photographes ne perturbent pas le phénomène à observer (par exemple il n'est pas raisonnable de demander aux oiseaux d'arrêter leur vol pendant les prises de vue). Il faut aussi que la scène reconstituée ait "un sens"; reste à définir ce qu'est "avoir un sens" et déterminer la manière de procéder pour faire ces prises de vue.

Le but de cet article est de présenter une synthèse des différentes solutions apportées à ce problème, en fonction des hypothèses faites sur les communications. Dans la partie 2 sont définis plus formellement le modèle de calcul, l'état global et les hypothèses sur les communications. La partie 3 présente l'algorithme de base de Chandy et Lamport, qui a servi de modèle à beaucoup d'autres algorithmes, et qui suppose des canaux de communica-

tion fifos (c'est à dire sur lesquels l'ordre de réception des messages est le même que leur ordre d'émission). Les solutions les plus représentatives dans le cas où les hypothèses de communication sont plus faibles que dans le cas précédent sont présentées dans la partie 4; la partie 5 présente deux solutions lorsque les hypothèses de communication sont plus fortes (ordre causal sur l'ensemble des messages). La partie 6 conclue l'article. Outre la présentation uniforme et cohérente de différentes solutions au problème de l'état global, un des intérêts de cet article est de montrer la variété des situations (et des hypothèses) qui peuvent apparaître dans un contexte réparti. Il y a lieu de noter que ces solutions sont présentées dans le cas du calcul d'un seul état global; le lecteur intéressé par le calcul d'une séquence d'états globaux pourra consulter [10].

2 Le modèle asynchrone

2.1 Le modèle de calcul

Un système réparti est constitué d'un ensemble fini de processus qui ne communiquent que par envoi de messages. La structure d'un tel système peut être modélisée par un graphe orienté: les sommets de ce graphe sont les processus et les arcs représentent les canaux de communication unidirectionnels (canaux virtuels) reliant des processus entre eux. A priori, aucune hypothèse particulière n'est faite quant à la topologie du graphe. La figure 1 montre un exemple de graphe de communication. Les processus seront désignés par P_1, P_2,..., P_n et le canal allant de P_i vers P_j, s'il existe, sera désigné par c_{ij}. Les processus ne partagent ni mémoire commune ni horloge globale, et aucune hypothèse n'est faite quant à leur vitesse relative. L'envoi et la réception de messages sont effectués de manière asynchrone, mais aucun délai maximum de transfert des messages n'est supposé connu. La seule hypothèse générale sur les communications concerne leur fiabilité: les messages ne sont ni perdus (le délai de transfert est fini: tout message émis est reçu) ni altérés ni dupliqués (tout message reçu a été émis). De telles hypothèses caractérisent ce qu'il est convenu d'appeler un système réparti asynchrone fiable.

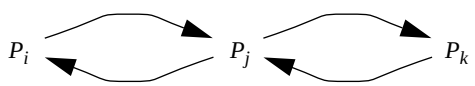

Figure 1. Un graphe de communication.

2.2 Etat global

Chaque processus et chaque canal possède à tout moment un état local. L'état local el_i d'un processus P_i résulte de son état initial et de la séquence d'événements dont ce processus a été le siège. L'état ec_{ij} d'un canal c_{ij} est l'ensemble des messages en transit sur ce canal, c'est à dire qui ont été émis par le processus P_i et n'ont pas encore été reçus par le processus P_j.

L'évolution du système est régie par un ensemble d'événements. Chaque événement met en jeu un processus et éventuellement un canal. Il y a trois sortes d'événements: les événements internes à un processus, qui ne modifient que l'état local du processus, les émissions de messages, qui modifient l'état local du processus émetteur et l'état du canal sortant sur lequel est émis le message, et les réceptions de messages, qui modifient l'état local du processus récepteur et l'état du canal entrant par lequel le message a été reçu. Schématiquement on a, à ce niveau de description (m désigne un message):

- événement interne sur P_i: provoque la transition de el_i à el'_i, états avant et après l'événement
- émission de m par P_i sur c_{ij} (cet événement est noté $émission_i(m)$): provoque la transition de el_i à el'_i et l'affectation $ec_{ij} := ec_{ij} \cup \{m\}$
- réception de m par P_i sur c_{ji} (cet événement est noté $réception_i(m)$): provoque la transition de el_i à el'_i et l'affectation $ec_{ji} := ec_{ji} \setminus \{m\}$.

Chacun de ces événements est supposé atomique. On dit qu'un événement e est capté dans l'état local el_i d'un processus P_i si e appartient à la séquence d'événements ayant conduit P_i dans l'état el_i. Il est important de remarquer que l'état local d'un processus n'est immédiatement observable que par un observateur local à ce processus (c'est à dire ayant accès en lecture à la mémoire locale de ce processus), tandis que l'état local d'un canal c_{ij} n'est immédiatement observable ni par son origine P_i ni par son extrémité P_j (P_i -resp. P_j- n'a pas connaissance immédiate des événements de réception -resp. d'émission-); c'est notamment ce fait qui rend difficile le problème de l'observation cohérente dans un système réparti.

L'état global S d'un système réparti est constitué de l'ensemble des états locaux des processus et des canaux qui le constituent:

$$S = \{\bigcup_i el_i\ ,\ \bigcup_{(i,j)} ec_{ij}\}$$

Un état global cohérent correspond à un état global dans lequel le système peut se trouver. Formellement, cela signifie:

i) $\forall i$: el_i est un état local du processus P_i.

ii) Les conditions $\mathbf{C_1}$ et $\mathbf{C_2}$ qui suivent sont vérifiées:

$\mathbf{C_1}$: si l'événement *émission*$_i(m)$ est capté dans el_i, alors l'événement *réception*$_j(m)$ est soit capté dans el_j, soit le message m appartient à ec_{ij}.

$\mathbf{C_2}$: si l'événement *émission*$_i(m)$ n'est pas capté dans el_i, l'événement *réception*$_j(m)$ n'est pas non plus capté dans el_j.

Un état global cohérent correspond à ce qu'il est convenu d'appeler une " coupe cohérente"(*consistent cut*). Le lecteur intéressé trouvera dans [5] une présentation formelle de la notion d'état global.

Exemple

Considérons l'exécution représentée par la figure 2, où trois processus P_i, P_j et P_k captent respectivement leurs états locaux el_i, el_j et el_k (le graphe de communication est celui de la figure 1).

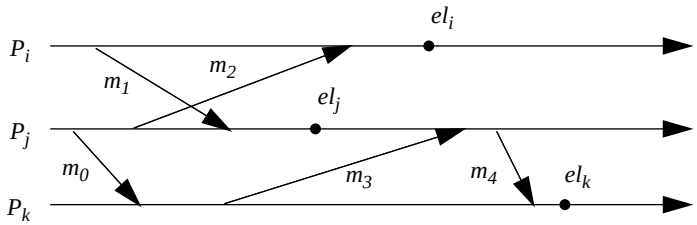

Figure 2. Une exécution répartie.

L'ensemble $\{el_i, el_j, el_k\}$ ne forme pas un état global. Si l'on considère la paire (P_k, P_j), l'émission du message m_3 est enregistrée dans l'état local el_k du processus P_k alors que sa réception ne l'est pas dans l'état local el_j du processus P_j. La condition $\mathbf{C_1}$ est mise en défaut, puisque l'état du canal ec_{kj} n'est pas considéré (un éventuel redémarrage de P_j et de P_k, suite à une reprise après défaillance, à partir de cet état global entraînerait la perte du message m_3). D'autre part, même en considérant l'état des canaux dans l'état global, la cohérence n'est pas assurée car la réception du message m_4 est enregistrée dans el_j sans que son émission le soit dans el_k, ce qui met en défaut la condition $\mathbf{C_2}$ (les deux états locaux el_j et el_k et l'état du canal ec_{jk} ne sont pas mutuellement cohérents).

2.3 Hypothèses sur les communications

Dans la partie 2.1, la seule hypothèse faite sur les communications est la fiabilité des canaux. Un certain nombre de solutions au problème du calcul d'un état global réparti s'appuient sur des hypothèses supplémentaires examinées ci-dessous. Afin de les exprimer précisément, nous rappelons la définition de la relation de causalité, due à Lamport [12] et notée $\rightarrow$, qui structure l'ensemble des événements produits par une exécution.

Définition (causalité): étant donné deux événements e et e' se produisant respectivement sur les processus P_i et P_j, $e \rightarrow e'$ est la plus petite relation satisfaisant les trois points suivants:

i) P_i=P_j et e est avant e' dans la séquence d'événements de P_i
ii) e est l'émission d'un message m par P_i (événement $émission_i(m)$), et e' laréception de ce même message par P_j (événement $réception_j(m)$)
iii) il existe e'' tel que $e \rightarrow e''$ et $e'' \rightarrow e'$ (fermeture transitive).

2.3.1 Hypothèses propres à un canal

Soit c un canal, m et m_1 deux messages empruntant ce canal. (Lorsque l'émetteur ou le récepteur d'un message n'importe pas, on n'indique pas son identité en indice de l'événement correspondant.)

On dira que m double m_1 si, et seulement si: $émission(m_1) \rightarrow émission(m)$ et $réception(m) \rightarrow réception(m_1)$

Quatre types possibles de messages peuvent être définis selon les contraintes de doublement [2].

i) Un message m est de type *marqueur*, s'il ne peut ni doubler ni être doublé par aucun autre message transitant par le même canal (figure 3):
$\forall m_1$ $émission(m_1) \rightarrow émission(m) \Rightarrow réception(m_1) \rightarrow réception(m)$
et $\forall m_2$ $émission(m) \rightarrow émission(m_2) \Rightarrow réception(m) \rightarrow réception(m_2)$

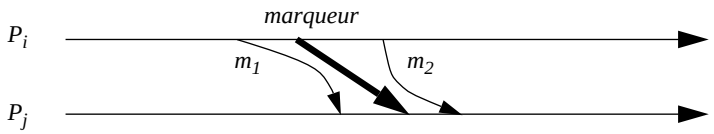

Figure 3. Livraison d'un marqueur.

ii) Un message m est de type *ct-passé* (contraint par son passé), s'il ne peut doubler aucun message (figure 4):

$\forall m_1$: *émission*(m_1) $\rightarrow$ *émission*(m) $\Rightarrow$ *réception*(m_1) $\rightarrow$ *réception*(m)

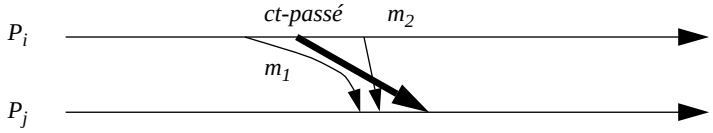

Figure 4. Livraison d'un message contraint par son passé.
(tous les messages émis avant lui sont reçus avant lui)

iii) Un message m est de type *ct-futur* (contraint par son futur), s'il ne peut être
doublé par aucun message (figure 5):
$\forall m_2$: *émission*(m) $\rightarrow$ *émission*(m_2) $\Rightarrow$ *réception*(m) $\rightarrow$ *réception*(m_2)

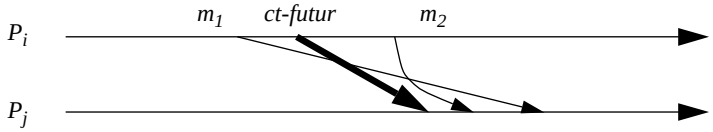

Figure 5. Livraison d'un message contraint par son futur.
(tous les messages émis après lui seront reçus après lui)

iv) Un message m de type *ordinaire* n'impose pas de condition de réception, mais respecte celles imposées par les autres messages transmis sur le même canal que lui (il ne peut doubler les messages *marqueur* et *ct-futur* et
ne peut être doublé par les messages *marqueur* et *ct-passé*).

En fonction des types des messages émis sur un canal, toute une gamme de comportements peut être définie pour ce canal, depuis le moins contraint (tous les messages sont *ordinaires*) jusqu'au plus contraint (tous les messages sont de type *marqueur*). Dans ce dernier cas, on dit que le canal est fifo, car les messages sont alors reçus dans l'ordre où ils ont été émis. De plus, pour un canal donné, les trois propriétés $\mathbf{A_1}$, $\mathbf{A_2}$ et $\mathbf{A_3}$ suivantes sont équivalentes [2]; A_1: tous les messages sont de type *ct-passé*; A_2: tous les messages sont de type *ct-futur*; A_3: tous les messages sont de type *marqueur*.

L'algorithme de base dû à Chandy et Lamport [7], exposé dans la partie 3, suppose que tous les canaux sont fifos. A l'opposé, les deux algorithmes de Lai et Yang [11] et de Mattern [13], qui font l'objet de la partie 4, ne nécessitent aucune hypothèse particulière, c'est à dire fournissent une solution lorsque tous les messages sont *ordinaires*. Les solutions dues à Ahuja [3] utilisent différents types de messages sur un même canal.

2.3.2 Hypothèses globales sur les communications

Les contraintes précédentes sont relatives à un canal donné. La propriété d'ordre causal porte sur l'ensemble des canaux. On dit que l'ensemble des messages du système satisfait la propriété d'ordre causal [6,14,15,18] si:

$\forall P_i, \forall P_j, \forall P_k$, $\forall m$ émis sur c_{ij}, $\forall m_1$ émis sur c_{kj}:

$$émission_i(m) \rightarrow émission_k(m_1) \Rightarrow réception_j(m) \rightarrow réception_j(m_1)$$

La figure 6 illustre un flot de messages qui ne respecte pas l'ordre causal (m_1 devrait être délivré après m pour avoir un tel ordre). On peut remarquer que si la propriété d'ordre causal est garantie alors chaque canal a un comportement fifo. Par contre, le fait que chaque canal soit fifo n'implique pas l'ordre causal sur l'ensemble des messages.

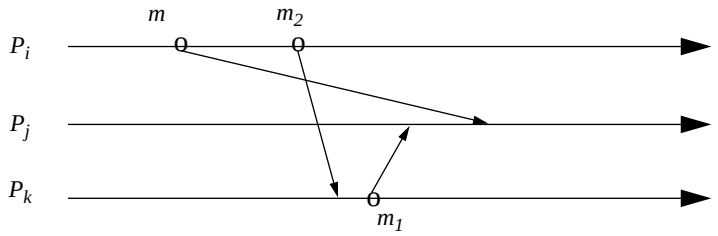

Figure 6. Réception non causale.

L'ordre causal définissant un ordre sur les réceptions de messages strictement plus fort que celui imposé par des canaux fifos, on peut s'attendre à avoir des solutions plus " simples"que celle de Chandy et Lamport lorsque le réseau de communication garantit cette propriété [17]. La partie 5 présente les algorithmes d'Acharya et Badrinath [1] et d'Alagar et Venkatesan [4], qui supposent qu'une telle propriété est réalisée. Des protocoles qui implémentent cette propriété sont décrits dans [6,14,15,18].

2.4 Superposition

A chaque processus P_i est associé un processus observateur CTL_i; ce processus peut lire l'état de P_i. La réception d'un message par P_i est réalisée par CTL_i, qui le délivre ensuite à P_i; de même, P_i confie à CTL_i les messages qu'il veut transmettre vers tout autre processus P_j; CTL_i se charge alors de son émission vers CTL_j (qui le délivre ensuite à P_j). Les processus CTL_i sont chargés de capter un état global cohérent de l'application. Ils coopèrent entre eux à l'aide de messages de contrôle. Un tel schéma d'observation des processus P_i par des contrôleurs CTL_i est appelé superposition [8]. Par abus de langage, on confond parfois P_i et son observateur CTL_i, lorsque ceci n'altère pas la compréhension.

3 Solution de Chandy et Lamport

3.1 Principe

L'article de Chandy et Lamport [7] est le premier, historiquement, à avoir correctement posé et résolu le problème du calcul d'un état global cohérent. Il fonctionne sous l'hypothèse de canaux fifos. Dans ces conditions, l'état d'un canal peut être représenté par une structure plus riche qu'un ensemble, à savoir une file. Lorsqu'un message est émis par P_i sur le canal c_{ij}, il est déposé en queue de la file ec_{ij}. La réception d'un message par P_j sur le canal c_{ij} correspond au retrait de l'élément de tête de la file ec_{ij}. Autrement dit, l'ensemble des messages envoyés sur le canal c_{ij} est totalement ordonné par la relation $m_1 < m_2 \Leftrightarrow$ *émission*$(m_1) \rightarrow$ *émission*$(m_2) \Leftrightarrow$ *réception*$(m_1) \rightarrow$ *réception*(m_2). Ainsi, tout message m de cet ensemble sépare l'ensemble en deux sous-ensembles disjoints, celui constitué des messages émis avant m, noté $<_m$, et celui constitué des messages émis après m, noté $>_m$. Cette propriété des canaux fifos est utilisée dans l'algorithme.

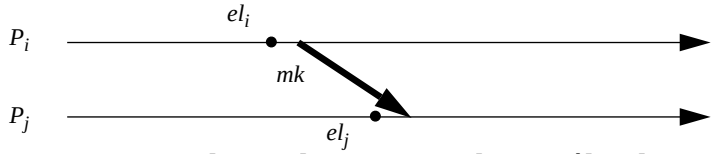

Figure 7. Echange d'un message de contrôle mk.

Considérons l'enregistrement de l'état local el_i du processus P_i (figure 7). Seules les émissions de messages antérieures à cet événement sont captées dans el_i. Donc, pour assurer la cohérence d'un couple d'états locaux (el_i, el_j) (conditions $\mathbf{C_1}$ et $\mathbf{C_2}$), seules les réceptions de messages émis par P_i avant l'enregistrement de el_i doivent être captées dans el_j. Pour mettre en oeuvre cette contrainte, le processus P_i émet, lorsqu'il enregistre son état el_i, un message de contrôle (appelé *mk*) sur le canal c_{ij}. Le processus P_j doit alors enregistrer son état local el_j au plus tard à la réception de ce message (*mk*).

En effet, par définition

$<_{mk} = \{m \mid$ *émission*$_i(m)$ captée dans $el_i\}$

et $>_{mk} = \{m \mid$ *émission*$_i(m)$ non captée dans $el_i\}$

De plus, el_j étant enregistré au plus tard à la réception du message *mk*

$<_{mk} \supseteq \{m \mid$ *réception*$_j(m)$ captée dans $el_j\}$

d'où:

$\{m \mid$ *réception*$_j(m)$ captée dans $el_j\} \subseteq \{m \mid$ *émission*$_i(m)$ captée dans $el_i\}$, ce qui assure le respect des conditions $\mathbf{C_1}$ et $\mathbf{C_2}$.

Cette propriété -évidente- est à la base de l'algorithme de Chandy et Lamport qui peut informellement être décrit de la manière suivante. Chaque processus P_j enregistre son état local el_j et l'état de ses canaux entrants en observant les deux règles suivantes:

R_1: un processus P_j qui capte son état local envoie sur tous ses canaux de sortie un message de contrôle (appelé *mk*) pour signifier à ses voisins qu'il a enregistré son état local.

R_2: à la réception d'un message *mk* sur un de ses canaux d'entrée, par exemple c_{ij}, un processus P_j peut se trouver dans l'une des deux situations suivantes:

1. P_j n'a pas encore capté son état local. Il doit alors enregistrer cet état, selon la règle **R_1**. L'état ec_{ij} du canal c_{ij} est vide car P_j a reçu tous les messages émis avant le message *mk* (les canaux sont fifos) et ceux émis après le message *mk* ne font pas partie de l'état global.

2. P_j a déjà enregistré son état local (soit de sa propre initiative, soit suite à une réception antérieure d'un message *mk* sur un autre canal d'entrée). L'état du canal relativement au couple d'états locaux el_i et el_j est constitué des messages reçus sur ce canal après que P_j ait enregistré son état local et avant qu'il ne reçoive *mk* (leurs émission est captée dans el_i, mais leur réception ne l'est pas dans el_j).

Il est intéressant de remarquer que, dans un système disposant d'une horloge physique globale, le problème serait facilement résolu: il suffirait de décider d'une date d à laquelle tous les processus du système enregistreront leur état local. L'identification des messages en transit se ferait aisément en associant à chaque message la date de son émission. Ceux qui sont émis avant la date d et reçus après la date d sont les messages en transit et sont donc enregistrés dans l'état du canal c_{ij} par P_j. Dans un système réparti à canaux fifos, les messages *mk* servent à simuler l' occurrence de la date d, et délimitent sur chaque canal les messages émis avant et après cette date.

Une description formelle de l'algorithme, empruntée à [16], est donnée dans la partie 3.2 ci-dessous.

3.2 Description formelle

Chaque processus P_j du calcul observé est doté de k_j canaux d'entrée et de l_j canaux de sortie; il est ainsi pourvu des déclarations suivantes:

c_in_j : [1..k_j] canaux d'entrée;
c_out_j : [1..l_j] canaux de sortie

l'observateur CTL_j associé à ce processus P_j, nécessaire au calcul de l'état global, a accès à ces canaux et est de plus doté des variables suivantes:

el_j : état local;
ec_j : [1..k_j] files de messages;
$reçu_j$: [1..k_j] booléens initialisés à faux;
$enreg\_état_j$: booléen initialisé à faux

el_j et $ec_j[x]$ vont servir à mémoriser respectivement l'état local et l'état du canal $c\_in_j[x]$; le booléen $reçu_j[x]$ sert de drapeau qui passe à *vrai* lorsqu'un message *mk* est reçu sur le canal $c\_in_j[x]$; enfin
$enreg\_état_j$ sert à indiquer que le processus P_j a enregistré son état local.
La procédure suivante est exécutée par CTL_j lorsque l'état local de P_j est capté.

procédure *capter_état_local*
début
el_j := *état local du processus* P_j;
$enreg\_état_j$:= *vrai*;
$\forall x : 1 \le x \le k_j : ec_j[x] := \emptyset$;
$\forall y : 1 \le y \le l_j$: **envoyer** *mk* **sur** $c\_out_j[y]$
fin

La procédure *fini* est exécutée par CTL_j lorsque la participation de P_j à l'état global est calculée: il a alors calculé son état local et celui de tous ses canaux entrants.

procédure *fini*
début
si $\forall x : 1 \le x \le k_j$: $reçu_j[x]$ **alors** *%la participation de* P_j *est calculée*
stocker (el_j, ec_j[1..k_j]) *chez le processus prévu à cet effet*
fsi
fin

Le comportement de CTL_j est exprimé par les deux énoncés suivants. Le premier énoncé est exécuté par un nombre quelconque de processus, au moins un, au plus n.

```
lors de la décision locale de calculer un état global
    début
        si non enreg_état_j alors capter_état_local fsi
    fin
lors de la réception de m sur c_in_j[x]
    début
        si m=mk
            alors   si non enreg_état_j alors capter_état_local
                    fsi;
                    reçu_j[x]:=vrai;
                    fini;
            sinon   si enreg_état_j et non reçu_j[x]
                        alors ec_j[x]:=ec_j[x]⊕m
                                %ajout de m en queue de ec_j[x]%
                    fsi;
                    délivrer m au processus P_j du calcul observé
        fsi
    fin
```

4 Solutions pour les canaux non fifos

Dans le problème qui nous intéresse, il est essentiel d'exiger d'un algorithme d'observation que son exécution ait aussi peu d'influence que possible sur l'ordre de réception des messages de l'exécution observée [19].

L'utilisation de messages de contrôle, telle qu'elle est prévue dans l'algorithme précédent, est élégante puisqu'elle permet de séparer l'ensemble des messages émis avant l'enregistrement de l'état local du processus émetteur de l'ensemble de ceux émis après (si les canaux n'étaient pas fifos, les messages *mk* seraient en fait des *marqueurs*). Si les canaux ne sont plus supposés fifos, la délimitation ne peut plus se faire aussi facilement sans perturber l'ordre des messages. C'est pourquoi d'autres techniques, n'utilisant pas de messages de contrôle explicites, ont été d'abord proposées: la partie 4.1, ci-après, examine deux solutions (Lai et Yang [11], Mattern[13]). Cependant, plus récemment, Ahuja [3] a donné une solution utilisant des messages de contrôle, ayant la simplicité de la solution de Chandy et Lamport tout en respectant la contrainte de non perturbation (partie 4.2).

4.1 Solutions sans messages de contrôle explicite

4.1.1 Méthode cumulative (mais rapide) de Lai et Yang

Cet algorithme [11] est basé sur l'utilisation de deux couleurs, symbolisant, pour les processus, l'avant (vert) et l'après (rouge) enregistrement de l'état local et, pour les messages, la couleur de leur émetteur. Chaque processus obéit aux règles suivantes:

R_1: Un processus qui n'a pas encore enregistré son état est coloré en vert, et tous les messages qu'il émet sont colorés en vert.

R_2: Lorsqu'un processus enregistre son état local, il devient rouge et tous les messages qu'il émet ensuite sont colorés en rouge.

R_3: Un processus vert peut à tout instant enregistrer son état local, et au plus tard lorsqu'il reçoit un message rouge.

R_4: Les messages verts émis et reçus sur chaque canal sont stockés (aspect cumulatif de l'algorithme).

R_5: Chaque processus rouge transmet à un même collecteur son état local et l'ensemble des messages (verts) qu'il a stockés.

Un état global cohérent est calculé par le processus collecteur lorsque ce dernier a reçu les informations de tous les processus participants; l'état ec_{ij} d'un canal c_{ij} est la différence entre l'ensemble des messages verts émis par P_i, vers P_j, noté $msg\_émis_i[j]$, et l'ensemble des messages verts reçus par P_j depuis P_i, noté $msg\_reçus_j[i]$. Il est facile de voir que les deux conditions $\mathbf{C_1}$ et $\mathbf{C_2}$ sont bien vérifiées; soit m un message ayant traversé le canal c_{ij}.

- Si l'émission de m est captée dans el_i, alors m est coloré en vert ($\mathbf{R_1}$) et $m \in msg\_émis_i[j]$ ($\mathbf{R_4}$)
  si sa réception est captée dans el_j, alors $m \in msg\_reçus_j[i]$ ($\mathbf{R_4}$) et donc $m \notin ec_{ij}$
  si sa réception n'est pas captée dans el_j, alors $m \notin msg\_émis_i[j]$ ($\mathbf{R_4}$) et donc $m \in ec_{ij}$ ceci assure la condition $\mathbf{C_1}$.
- Si l'émission de m n'est pas captée dans el_i, alors m est coloré en rouge ($\mathbf{R_2}$). Lorsque m est reçu par P_j, ce dernier processus a déjà enregistré son état local, ou doit le faire sans prendre en compte m ($\mathbf{R_3}$). Donc la réception de m n'est pas captée dans el_j. Ceci assure la condition $\mathbf{C_2}$.

Par rapport à l'algorithme de Chandy et Lamport, aucun message de contrôle (supplémentaire) n'est nécessaire. Par contre, le processus observateur CTL_i doit observer les envois et réceptions de messages effectués par P_i de

manière continue (ce qui n'était pas le cas dans les algorithmes adaptés aux canaux fifos), et le volume d'information à stocker pour chaque processus est important (messages verts). Ce dernier inconvénient (aspect cumulatif) disparaît dans l'algorithme de Mattern [13] qui utilise des compteurs et des messages supplémentaires pour éviter le stockage des ensembles de messages; ceci est obtenu au prix d'un plus grand délai avant l'obtention de l'état global résultat.

4.1.2 Méthode non cumulative (mais lente) de Mattern

L'algorithme [13] est fondé sur la propriété suivante: les messages en transit sur un canal c_{ij} sont exactement les messages verts émis par P_i (captés dans el_i), et reçus par le processus rouge P_j (non captés dans el_j). Le processus P_j sait que les messages verts qu'il a reçus après son enregistrement d'état local sont les messages en transit sur ce canal. Par contre, il ne peut pas savoir si tous les messages verts qui lui sont destinés lui sont parvenus, puisqu'un message rouge peut doubler un message vert. L'utilisation de compteurs locaux, combinés globalement par un processus collecteur, permet de résoudre ce problème. Supposons que chaque processus P_i enregistre, avec son état local, la différence d_i entre le nombre de messages qu'il a émis avant l'enregistrement de son état local (messages verts émis) et le nombre de messages verts reçus avant l'enregistrement de son état local. Il est facile de voir que le nombre $mt = \sum d_i$ est exactement le nombre total de messages en transit relativement à l'état global (el_1,..., el_n) enregistré. De ces deux propriétés découle l'algorithme.

La règle de coloration des processus et des messages, ainsi que les règles $\mathbf{R_1}$ à $\mathbf{R_3}$ sont identiques à celles de l'algorithme de Lai et Yang. Les règles $\mathbf{R_4}$ et $\mathbf{R_5}$ sont remplacées par les suivantes:

$\mathbf{R'_4}$: Chaque processus P_i compte la différence d_i entre le nombre de messages verts qu'il a émis et le nombre de messages verts qu'il a reçus.

$\mathbf{R'_5}$: Chaque processus rouge transmet au collecteur son état local et la valeur de son compteur d_i.

$\mathbf{R'_6}$: Un processus rouge transmet au collecteur les messages verts qu'il reçoit.

Un état global cohérent est calculé par le processus collecteur de la manière suivante:

- dans un premier temps, lorsque le collecteur a reçu les informations de tous les processus participants, il initialise un compteur mt à la valeur $\sum d_i$.

- ensuite, chaque message retransmis au collecteur par un des processus participants (messages en transit, cf. $\mathbf{R'_6}$) est ajouté à l'état du canal correspondant, et provoque le décrément du compteur *mt*.
- lorsque *mt*=0 le collecteur conclue qu'il a obtenu l'état de tous les canaux.

Par rapport à l'algorithme de Lai et Yang, il y a un nombre de messages supplémentaires égal au nombre de messages en transit (donc a priori fini mais non borné), et de plus, il faut attendre que tous les messages en transit soient reçus pour connaître l'état global (c'est en ce sens que cet algorithme est plus " lent"que celui de Lai et Yang).

Remarque: l'algorithme de Chandy et Lamport peut, lui aussi, être exprimé à l'aide des couleurs (Dijktsra [9]): il se réduit alors aux règles $\mathbf{R_1}$, $\mathbf{R_2}$ et $\mathbf{R_3}$ des deux algorithmes précédents, complétées par la règle $\mathbf{R''_4}$: les messages en transit sur le canal c_{ij} sont les messages verts reçus par le processus rouge P_j. Le premier message rouge traversant le canal c_{ij} joue le rôle de message de contrôle (*marqueur*) de l'algorithme de Chandy et Lamport.

4.2 Solutions utilisant des messages de contrôle

Pour que l'algorithme de Chandy et Lamport soit correct, il n'est en fait pas nécessaire que les canaux soient fifos; la propriété essentielle concerne les messages de contrôle qui, eux, doivent être du type *marqueur* afin de jouer le rôle de séparateur. La figure 8 illustre ce phénomène: les messages m et m_1 se doublent. L'introduction d'un *marqueur* émis entre les deux ne peut se faire sans provoquer une perturbation dans l'ordre de réception de m et m_1.

Figure 8. Perturbation due au marqueur.

Il est clair que le rajout de messages de contrôle de type *ct-futur* ou *ct-passé* peut être effectué sans modifier l'ordre de réception des autres messages du système (figure 9).

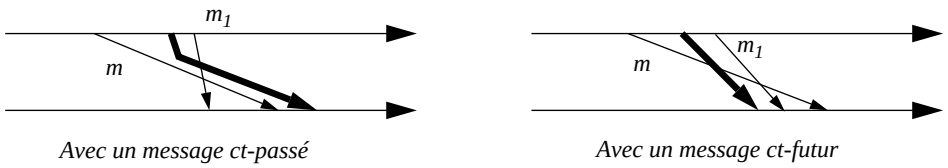

Figure 9. Pas de perturbation.

Avec un message *ct-futur*, la réception de m_1 peut éventuellement être retardée:

$$\textit{émission}_i(m) \rightarrow \textit{émission}_i(\textit{ct-futur}) \rightarrow \textit{émission}_i(m_1)$$
$$\text{et} \quad \textit{réception}_j(\textit{ct-futur}) \rightarrow \textit{réception}_j(m_1) \rightarrow \textit{réception}_j(m)$$

L'algorithme d'Ahuja [3] est informellement décrit de la manière suivante. Chaque processus P_j enregistre son état local el_j et l'état de ses canaux entrants en observant les règles $\mathbf{R_1}$ à $\mathbf{R_4}$ suivantes:

i) Règles de coordination des enregistrements d'états locaux:

$\mathbf{R}_1$: Un processus P_j qui enregistre son état local envoie sur tous ses canaux de sortie un message *enreg_état_local*, de type *ct-futur*, avant d'envoyer tout autre message.

$\mathbf{R_2}$: A la réception d'un message *enreg_état_local* sur un de ses canaux d'entrée, le processus P_j enregistre son état local selon la règle $\mathbf{R_1}$, s'il ne l'a pas déjà fait.

Ces deux règles permettent de capter un ensemble d'états locaux cohérents (condition $\mathbf{C_2}$). Si le message m, traversant le canal c_{ij}, est tel que $\textit{émission}_i(m)$ n'est pas capté dans el_i, alors m est émis après le message *enreg_état_local* sur le canal c_{ij}; m est donc reçu après ce message de contrôle, et donc $\textit{réception}_j(m)$ n'est pas capté dans el_j.

ii) Règles pour le calcul de l'état des canaux: les messages du système observé sont estampillés avec des numéros de séquence: le message du canal c_{ij} estampillé t, est le $(t+1)^{\text{ieme}}$ message envoyé sur ce même canal.

$\mathbf{R_3}$: Un processus P_j qui enregistre son état local envoie, sur tous ses canaux de sortie, après le message *enreg_état_local* et avant d'envoyer tout autre message, un message *enreg_état_canal* de type *ct-passé*, avec le même numéro que le dernier message émis.

$\mathbf{R_4}$: A la réception d'un message *enreg_état_canal*, numéroté t, sur un canal d'entrée c_{ij}, le processus P_j enregistre ec_{ij} comme étant l'ensemble des messages de numéro inférieur ou égal à t, reçus après l'enregistrement de l'état local el_j.

Ces deux règles permettent bien de capter l'état des canaux (condition $\mathbf{C_1}$). En effet, soit un message traversant le canal c_{ij}, en transit relativement aux états locaux el_i et el_j. Puisque $\textit{émission}_i(m)$ est captée dans el_i, le numéro de ce message est inférieur ou égal à t. Puisque $\textit{réception}_j(m)$ n'est pas capté dans el_j, m est reçu par P_j après l'enregistrement de son état local. Enfin, puisque le message *enreg_état_canal* est de type *ct-passé*, il ne peut pas dou-

bler. Donc:

$$\textit{émission}_i(m) \rightarrow \textit{émission}_i(\textit{enreg\_état\_canal}) \Rightarrow \textit{réception}_j(m) \rightarrow \textit{réception}_j(\textit{enreg\_état\_canal})$$

D'après la règle $\mathbf{R_4}$, m est bien enregistré dans ec_{ij}.

Réciproquement, si m est enregistré dans ec_{ij}, il est reçu par P_j après l'enregistrement de l'état local el_j et, puisqu'il est de numéro inférieur ou égal à t, il a été émis par P_i avant l'enregistrement de l'état local el_i. C'est donc bien un message en transit relativement aux états locaux el_i et el_j.

Cette solution, qui reste fidèle à " l'esprit" de Chandy et Lamport, nécessite un nombre de messages de contrôle double, ainsi que la numérotation des messages. Mais elle évite un inconvénient inhérent aux algorithmes de Lai et Yang et de Mattern, à savoir la nécessité de transmettre toutes les informations nécessaires au calcul de l'état global à un processus coordinateur. Par rapport à ces deux algorithmes, la solution d'Ahuja conserve la caractère décentralisé rencontré chez Chandy et Lamport.

5 Solutions fondées sur l'ordre causal

On suppose maintenant que le réseau de communication sur lequel s'exécute le programme observé, garantit l'ordre causal défini dans la partie 2. Des protocoles de communication permettant d'inplémenter cette contrainte sont décrits dans [6,14,15,18]: les structures de données utilisées par de tels protocoles peuvent être utilisées (en lecture) par les protocoles de capture d'état global (voir en 5.2.1). Ci-dessous sont présentés deux algorithmes tirant profit de cette hypothèse sur les communications, qui, tous deux, améliorent l'algorithme de Chandy et Lamport, en ce sens qu'ils ne nécessitent pas de messages de contrôle supplémentaires sur tous les canaux mais seulement d'un processus vers tous les autres.

Les deux algorithmes ont en commun les règles relatives à l'envoi et à la réception des messages de contrôle, permettant de coordonner l'enregistrement des états locaux. Ils diffèrent dans la manière de calculer l'état des canaux: calcul centralisé, dans le cas du premier algorithme, décentralisé dans le cas du second. Dans l'exposé qui suit, un seul processus joue le rôle d'initiateur (cette restriction peut toutefois être partiellement levée dans le cas du second algorithme).

5.1 Calcul des états locaux

R_1: L'initiateur P_{init} diffuse à tous les autres (y compris à lui même) un message de contrôle *enreg_état_local*.

R_2: Lorsqu'un processus P_i reçoit ce message de contrôle, il enregistre son état local el_i.

Ces deux règles assurent que l'ensemble des états locaux enregistrés est cohérent (condition **C_2**). En effet, considérons un message m envoyé sur le canal c_{ij}, tel que l'événement $émission_i(m)$ n'est pas capté dans el_i. Il a donc été émis par P_i après que ce dernier ait reçu le message de contrôle *enreg_état_local*, donc:

$$émission_{init}(enreg\_état\_local) \rightarrow réception_i(enreg\_état\_local) \rightarrow émission_i(m)$$

si sa réception était captée dans el_j, on aurait:

$$réception_j(m) \rightarrow réception_j(enreg\_état\_local)$$

et l'hypothèse d'ordre causal serait violée.

5.2 Calcul de l'état des canaux

5.2.1 Méthode centralisée d'Acharya et Badrinath

Dans cet algorithme [1], chaque processus P_i gère deux tableaux $sent_i[1..n]$ et $recd_i[1..n]$ qui servent à compter les messages émis et reçus par P_i:

$$sent_i[j]=\alpha \Leftrightarrow P_i \text{ a envoyé } \alpha \text{ messages à } P_j$$
$$recd_i[j]=\beta \Leftrightarrow P_i \text{ a reçu } \beta \text{ messages de } P_j$$

Les deux tableaux étant déjà utilisés dans la plupart des protocoles qui réalisent l'ordre causal [15], leur utilisation par le protocole décrit n'est pas pénalisante.

La règle **R_2** est complétée de la manière suivante:

R'_2: Lorsqu'un processus P_i reçoit le message *enreg_état_local*, il enregistre son état local el_i, et les deux tableaux $sent_i$ et $recd_i$, puis envoie à l'initiateur le message $état\_local(el_i, sent_i, recd_i)$.

La règle suivante est observée par l'initiateur:

R_3: Lorsque P_{init} a reçu tous les messages *état_local*, il possède tous les états locaux el_i et calcule l'état des canaux de la manière suivante:

$\forall j$ $\quad c_{init,j} := \emptyset$

$\forall i \neq init, \forall j$: $\quad c_{ij} := \{recd_j[i]+1,...,sent_i[j]\}$ (ce sont tous les numéros de messages que P_i a envoyés à P_j et que P_j n'a pas encore reçu de P_i).

L'état des canaux ainsi calculé est correct. En, effet, soit m le t^{ieme} message émis par P_i vers P_j; c'est aussi le $t^{ième}$ messages qui est -ou sera- reçu par P_j depuis P_i (ordre causal):

$m \in c_{ij} \quad \Leftrightarrow \quad recd_j[i] < t \leq sent_i[j]$
$\Leftrightarrow$ *réception*$_j(m)$ non captée dans el_j et *émission*$_i(m)$ captée dans el_i
$\Leftrightarrow$ m en transit relativement aux états locaux el_i et el_j.

5.2.2 Méthode répartie d'Alagar et Venkatesan

Dans cet algorithme [4], et contrairement au précédent, chaque processus est chargé du calcul de l'état de ses canaux entrants, comme dans les algorithmes de Chandy et Lamport ou d'Ahuja. Pour cela, une technique de coloration des messages est utilisée, mais au lieu de colorer les messages -en vert ou en rouge- au moment de leur émission (ce qui oblige les messages à " transporter"cette couleur), la coloration est effectuée au moment de la réception: lorsqu'un processus P_j reçoit un message m émis pas P_i, ce dernier message est coloré en rouge si, et seulement si:

$$émission_{init}(enreg\_état\_local) \rightarrow émission_i(m)$$

Les structures de données, maintenues par les protocoles qui implémentent l'ordre causal permettent au processus récepteur P_j d'effectuer ce test (voir [4] pour les détails du test).

L'algorithme est alors décrit par les règles ci-dessous. La règle $\mathbf{R_1}$ est inchangée; la règle $\mathbf{R_2}$ est complétée de la manière suivante:

R"$_2$: Lorsqu'un processus P_i reçoit le message *enreg_état_local*, il enregistre son état local el_i, initialise l'état des canaux entrants à vide, et répond à l'initiateur avec un message *fait*.
La règle suivante est appliquée à tout processus:

R$_3$: Lorsque, après avoir enregistré son état local, P_j reçoit un message de l'application sur c_{ij}, il teste si ce message est rouge. S'il ne l'est pas, il le met dans l'ensemble ec_{ij}.

La fin de l'algorithme d'enregistrement découle des règles suivantes:

R$_4$: Lorsque l'initiateur a reçu un message *fait* de tous les autres processus, il diffuse un message *terminé*.

R$_5$: Lorsqu'un processus P_i reçoit un message *terminé* de l'initiateur, il termine l'enregistrement des canaux.

L'état des canaux ainsi enregistré est correct. En effet, soit m un message traversant le canal c_{ij}, en transit par rapport aux états locaux el_i et el_j. On a:

$émission_i(m) \rightarrow réception_i(enreg\_état\_local)$ (m capté dans el_i)
$réception_i(enreg\_état\_local) \rightarrow émission_i(fait)$ (règle $\mathbf{R''_2}$)
$émission_i(fait) \rightarrow réception_{init}(fait)$ (définition de la relation $\rightarrow$)
$réception_{init}(fait) \rightarrow émission_{init}(terminé)$ (règle $\mathbf{R_4}$)

d'où: $émission_i(m) \rightarrow émission_{init}(terminé)$ (transitivité de la relation $\rightarrow$).
D'après l'ordre causal:
$réception_j(m) \rightarrow réception_j(terminé)$.
D'autre part, $réception_j(enreg\_état\_local) \rightarrow réception_j(m)$ (m non capté dans el_j).
Enfin, m n'est pas coloré en rouge puisque $émission_i(m)$ est capté dans el_i.
Le message m est donc bien enregistré dans ec_{ij} (règles $\mathbf{R_3}$ et $\mathbf{R_5}$).
Réciproquement, si m est enregistré dans ec_{ij}, on a:
$réception_j(enreg\_état\_local) \rightarrow réception_j(m) \rightarrow réception_j(terminé)$, et m n'est pas coloré en rouge.
Donc (1) $réception_j(m)$ n'est pas capté dans el_j
(2) $émission_i(m)$ est capté dans el_i

6 Conclusion

Le calcul d'un état global est un paradigme (au même titre que l'exclusion mutuelle) des problèmes de contrôle rencontrés dans les systèmes répartis. Après avoir posé le problème, cette synthèse a présenté des algorithmes types qui permettent d'obtenir des états globaux cohérents. Chacun d'eux correspond à un contexte et à des hypothèses particulières sur les communications (canaux fifos, messages de contrôle particuliers, ordre causal). La connaissance de ces hypothèses et des solutions associées constitue un élément important pour qui veut maîtriser la difficulté des calculs et des systèmes répartis.

Bibliographie

[1] A. Acharya and B.R. Badrinath, *Recording Distributed Snapshots Based on Causal Order of Message Delivery*. Inf. Proc. Letters, Vol. 44, (1992), pp. 317-321.

[2] M. Ahuja, *Flush Primitives for Asynchronous Distributed Systems.* Inf. Proc. Letters, Vol. 34,2, (1990), pp. 5-12.

[3] M. Ahuja, *Global Snapshots for Asynchronous Distributed Systems with Non-FIFO Channels.* Tech, Rep. #CS92-268, Univ. of California, San Diego, (Nov. 1992), 7 p.

[4] S. Alagar and S. Venkatesan, *An Optimal Algorithm for Distributed Snapshots with Causal Message Ordering.* Tech, Rep., Univ. of Texas, Dallas, Texas, (1993), 7p.

[5] O. Babaoglu and K. Marzullo, *Consistent Global States of Distributed Systems: Fondamental Concepts and Mechanisms.* In S.J. Mullender, Editor, Distributed Systems, Chap. 4, ACM Press, 1993.

[6] K. Birman and T.A. Joseph *Reliable Communication in the Presence of Failures.* ACM TDCS, Vol. 5,1, (Feb. 1987), pp. 47-76.

[7] K.M. Chandy and L. Lamport, *Distributed Snapshots: Determining Global States of Distributed Systems.* ACM TOCS, Vol. 3,1,(1985), pp. 63-75.

[8] K.M. Chandy and J. Misra. *Parallel Program Design: a Foundation.* Addison-Wesley, (1988), 516 p.

[9] E.W. Dijkstra, *The Distributed Snapshot of K.M. Chandy and L. Lamport.* Tech, Rep. EWD864a, Univ. of Texas, Austin, Tex., (1984).

[10] J.-M. Hélary, *Observing Global States of Asynchronous Distributed Applications.* Proc. Workshop on Dist. Alg., Springer Verlag, LNCS 392, (1989), pp. 124-135.

[11] T.H. Lai and T.H. Yang, *On Distributed Snapshots.* Inf. Proc. Letters, Vol. 25, (1987), pp. 153-158.

[12] L. Lamport, *Time, Clocks and the Ordering of Events in a Distributed System.* Comm. ACM, Vol. 21,7, (july 1978), pp. 558-565.

[13] F. Mattern, *Virtual Time and Global States of Distributed Systems.* Proc. of Int. Workshop on Parallel and Dist. Systems, North-Holland, 1988, pp. 215-226.

[14] L.L. Peterson, N.C. Bucholz and R. Schlichting, *Preserving and Using Context Information in Interprocess Communication.* ACM TOCS, Vol. 7,3, (1989), pp. 213-246.

[15] M. Raynal, A. Schiper and S. Toueg, *The Causal Order Abstraction and a Simple Way to Implement it.* Inf. Proc. Letters, Vol. 39, (1991), pp. 343-350.

[16] M. Raynal, *Synchronisation et état global dans les systèmes répartis.* Eyrolles, Collection EDF, (1992), 230 p.

[17] A. Sandoz and A. Schiper, *A Characterization of Consistent Distributed Snapshot Using Causal Order*. Tech. Report RR 92-14, EPLF, Lausanne, (Oct. 1992), 9 p.

[18] A. Schiper, A. Sandoz and J. Eggli, *A New Algorithm to Implement Causal Ordering.* Proc. Workshop on Dist. Alg., Springer Verlag, LNCS 392, (1989), pp. 219-232.

[19] K. Taylor. *The Role of Inhibition in Asynchronous Consistent Cut Protocols.* Proc. Workshop on Dist. Alg., Springer Verlag, LNCS 392, (1989), pp. 280-291.