

# Précis de répartition

## Aspects algorithmiques (suite)

3ième Année Informatique et Mathématiques Appliquées

11 octobre 2004

### Table des matières

<b>1</b>	<b>Calcul d'un état global : prise de cliché</b>	<b>2</b>
1.1	Introduction . . . . .	2
1.2	Notion de coupure cohérente . . . . .	2
1.3	Un algorithme de prise de cliché (Chandy et Lamport) . . . . .	3
<b>2</b>	<b>Le problème du consensus</b>	<b>4</b>
2.1	Spécification du problème . . . . .	5
2.2	Les différents contextes de résolution . . . . .	5
2.3	Preuve de l'impossibilité du consensus en asynchrone . . . . .	5
2.4	Un algorithme simple . . . . .	7
2.5	Quelques problèmes voisins . . . . .	7
<b>3</b>	<b>Conclusion</b>	<b>7</b>

# 1 Calcul d'un état global : prise de cliché

## 1.1 Introduction

L'absence d'état global est une des caractéristiques essentielle (et malheureuse...) des systèmes répartis. Un site ou processus n'a qu'une connaissance approximative de l'état des autres sites ou processus puisque, seul, l'échange de messages permet d'obtenir des informations sur les partenaires distants (logiquement).

L'intérêt de capter un cliché global d'un calcul réparti est avant tout de pouvoir, en cas de défaillance, reprendre le calcul à partir d'un tel cliché. Il s'agit donc de définir des points de reprise potentiels.

Un grand nombre d'algorithmes ont donc été proposés sur le thème de l'évaluation d'états globaux cohérents d'un calcul réparti. L'objectif de ces algorithmes est d'arriver à collecter un ensemble d'états locaux aux processus afin d'obtenir un état global passé cohérent du calcul. Cette prise de cliché (snapshot) pose surtout un problème de cohérence des informations collectées incluant d'éventuels messages en transit.

La figure (1) montre cette difficulté : le processus collecteur capte un état local  $E_A$  du processus  $A$  pour lequel le message  $m$  n'a pas encore été envoyé, alors que l'état local  $E_B$  du processus  $B$  est postérieur à la réception de  $m$ . C'est ce genre d'incohérence qui peut par exemple, dans un algorithme de détection d'un état stable, provoquer une fausse détection. Dans cette exemple, s'il s'agit de détecter la terminaison d'un calcul diffusant par comptage des messages envoyés et reçus par chaque processus, l'état global incohérent collecté peut conduire à une fausse détection du fait de la prise en compte de la réception du message  $m$  sans avoir pris en compte son émission.

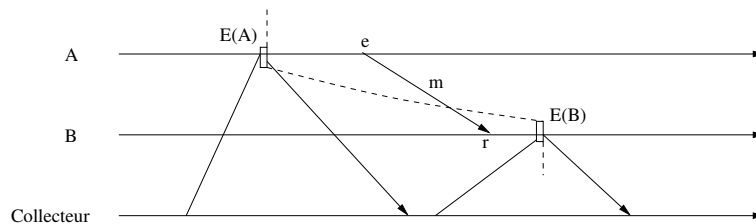


FIG. 1 – Prise de cliché incohérent

Une propriété de base à préciser est donc la notion de cohérence d'un état global. Pour ce faire, on utilise la notion de coupure cohérente.

## 1.2 Notion de coupure cohérente

De façon standard, le calcul réparti est abstrait sous la forme d'un ensemble d'événements partiellement ordonnés. On appelle coupure cohérente  $C$  un sous-ensemble des événements  $\mathcal{CR}$  d'un calcul réparti vérifiant la propriété suivante :

$$\forall e \in C : \forall e' \in \mathcal{CR} : e' \prec e \Rightarrow e' \in C$$

Autrement dit, si un événement  $e$  appartient à une coupure, alors tous les événements du calcul qui le précèdent causalement appartiennent aussi à la coupure. Une coupure, comme son nom le suggère, établit une frontière sur tous les sites entre un ensemble d'événements qui sont « avant » et la suite du calcul « après ». L'exemple de la figure (1) montre une coupure incohérente puisque l'événement d'émission  $e$  de  $m$  n'est pas dans l'ensemble des événements collectés qui contient, par contre, l'événement  $r$ .

Un algorithme de prise de cliché devra donc tout d'abord réaliser une coupure cohérente du calcul. Cependant, l'état global construit n'aura pas forcément existé dans le temps global. à titre d'exemple, la figure (2) montre une coupure cohérente bien qu'elle suppose un état global dans lequel l'événement  $e$  est arrivé et l'événement  $r'$  est encore à venir alors que dans la réalité l'événement  $r'$  a eu lieu avant  $e$ . Ceci n'a cependant pas d'importance puisque les deux événements ne sont pas causalement liés.

Par ailleurs, l'événement de réception du message  $m$  n'est pas capté dans la prise de l'état local du site  $B$ , alors que l'émission de ce message appartient au passé de l'état local capté du site  $A$ . Pour obtenir un

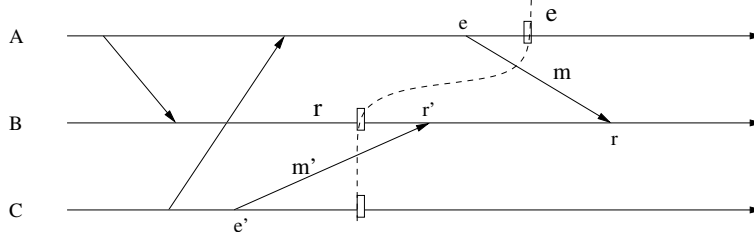


FIG. 2 – Prise de cliché cohérente mais d'un état potentiel

état cohérent, il faudra donc aussi mémoriser que le message  $m$  est un message en transit. Un état global cohérent associé à la coupure de la figure (2) devra donc aussi préciser que le message  $m$  est en transit.

### 1.3 Un algorithme de prise de cliché (Chandy et Lamport)

Cet algorithme assure la collecte d'un cliché cohérent constitué des états locaux de chaque site et des messages en transit associés. Pour ce faire, certaines hypothèses simplificatrices sont faites :

- la communication est supposée point à point sur des canaux FIFO unidirectionnels. Chaque site  $i$  connaît d'une part ses canaux entrants  $E_i$  et d'autre part ses canaux sortants  $S_i$  ;
- la topologie du réseau de communication doit être fortement connexe ; cette contrainte est due au principe de marquage utilisé.

Comme il n'est pas possible de synchroniser la prise de chaque état local à un instant  $t$ , la prise d'un cliché global est asynchrone. Chaque processus peut décider de débiter une phase d'évaluation d'un état local complété par d'éventuels messages en transit.

#### Phase d'évaluation d'un cliché local

Cette phase consiste à mémoriser son état local et à capter, à partir de cet instant, les messages en transit sur ses canaux entrants. Pour cela, une fois cet état local mémorisé, un message  $ij$  marqueur  $ij$  est émis sur chaque canal de sortie  $S_i$ . Après quoi, tout message entrant est mémorisé comme message en transit appartenant à un canal donné. Si un message marqueur arrive parmi ces messages entrant, alors tous les messages en transit sur ce canal ont été pris en compte. Lorsqu'un message marqueur aura été capté sur chaque canal entrant, l'état local mémorisé complété par les messages en transit de chaque canal entrant peut être envoyé au processus collecteur.

Lorsqu'un processus reçoit un message marqueur, deux cas sont possibles :

- soit il a déjà mémorisé son état local, auquel cas, il est en phase de réception des messages marqueurs et il n'a donc qu'à enregistrer que la collecte des messages en transit sur un canal entrant est terminée ;
- soit il n'a pas encore mémorisé son état local, auquel cas, il débute la phase d'évaluation d'un cliché local.

## 2 Le problème du consensus

Le problème du consensus est considéré par certains comme LE problème fondamental posé aux systèmes répartis pour les rendre tolérants aux fautes sous les hypothèses asynchrones. Il est en effet sous-jacent à toute situation où l'on utilise la redondance pour obtenir un système tolérant certaines fautes.

Par exemple, dans un système de gestion de transactions réparties, tous les processus ayant participé à une transaction doivent finalement décider de sa validation ou de son annulation. Ils doivent TOUS prendre la même décision. Dans les protocoles de diffusion, il faut pouvoir décider si un message a bien été reçu par tous les processus cibles. Chacun d'entre eux devra là encore prendre la même décision vis-à-vis d'un message donné. Lorsqu'on met en œuvre de la réplication, il faudra que chaque processus répliquant un calcul se mette d'accord avec les autres sur le résultat.

Le problème majeur du consensus est de tolérer les défaillances de sites ou de communication. Il faudra en particulier distinguer les processus corrects des processus défaillants. La défaillance même d'un processus pourra prendre plusieurs formes. Une forme simple est l'arrêt. Mais des comportements plus complexes pourront être considérés comme par exemple, au pire, un comportement arbitraire : un processus émet par exemple des messages erronés (voir problème des généraux byzantins). Pourtant, le problème est simple à spécifier : comment  $N$  sites (processus) peuvent se mettre d'accord, donc atteindre un consensus, en communiquant par messages ?

Sous ces apparences simples, le consensus se révèle un problème délicat qui ne doit pas être réduit à un simple problème d'acquiescement de message. Il suffit pour s'en convaincre d'un exemple comportant seulement deux interlocuteurs.

### Roméo et Juliette se donnent rendez-vous par e-mail

Supposons que Roméo souhaite donner rendez-vous à Juliette à une date et en un lieu fixé en utilisant l'e-mail et supposons que le service e-mail puisse perdre des messages.

Roméo envoie donc un premier message pour proposer un rendez-vous à Juliette. Tout ce passe bien et celle-ci le reçoit 5 mns plus tard. Elle répond oui par un message qui parvient 10mns plus tard à Roméo. Le consensus semble alors atteint entre Roméo et Juliette et ils devraient donc se rencontrer. Cependant, Roméo est alors pris d'un doute : Si Juliette ne sait pas que j'ai reçu sa réponse positive, elle peut supposer que son message réponse s'est perdu et peut-être n'ira-t-elle pas au rendez-vous. Je devrais donc lui envoyer un message confirmant que j'ai bien reçu sa réponse. Mais dans ce cas, une suite infernale d'acquiescements réciproques débute. En fait, sous ces hypothèses, il n'existe alors pas de protocole qui permette d'être sûr que les deux prendront la même décision.

Le scénario précédent amène les remarques suivantes :

- Même pour seulement deux processus, le problème est insoluble ;
- Le problème du consensus est strictement distinct de celui de l'acquiescement d'un message. Roméo reçoit bien un acquiescement de son premier message lorsque lui parvient la réponse de Juliette. Il sait alors que son premier message est bien parvenu à Juliette. Par contre, Juliette ne sait pas qu'il possède cette connaissance. Elle peut avoir un doute ! Le problème du consensus soulève donc un problème plus difficile de coordination mutuelle ;
- Si Roméo et Juliette avaient pris un téléphone, ils seraient parvenus à un consensus sans difficulté. En effet, la communication assure alors une borne supérieure à la délivrance ou à la perte d'un message (coupure de la ligne). C'est l'hypothèse d'un délai non borné de délivrance des messages qui rend le consensus impossible ;
- Une approche probabiliste peut être adoptée : Roméo et Juliette auraient pu dupliquer leurs messages de façon à minimiser le risque de perte d'un message. Il est alors intéressant de concevoir des algorithmes de consensus de type probabiliste garantissant qu'un consensus sera atteint dans un délai **borné** avec une **forte** probabilité. Ceci est possible car les situations rendant le protocole impossible à 100% sont rares.

## 2.1 Spécification du problème

Plus précisément, une modélisation du problème peut être faite sous la forme des hypothèses de base suivantes :

- le système est composé de  $N$  processus ;
- chaque processus possède une valeur initiale  $v_0 \in \mathcal{D}$  ;

Un algorithme correct doit vérifier les spécifications suivantes :

1. Terminaison : l'algorithme doit être vivace c'est-à-dire que tout processus correct doit finalement décider d'une valeur finale ;
2. Validité : si tous les processus ont initialement la même valeur  $v_0$ , tous les processus corrects choisiront finalement pour valeur finale  $v_0$  ;
3. Cohérence : si un processus correct choisit une valeur  $v$ , tous les autres processus choisiront aussi  $v$ .

Des variantes existent selon les contraintes de validité de la valeur finale : par exemple, la valeur finale doit être la valeur initiale possédée par une majorité de sites, voire par tous les sites. À titre d'exemple, on peut supposer que les processus doivent se mettre d'accord sur une valeur booléenne ( $\mathcal{D} = \text{Booléen}$ ). La contrainte de validité peut être alors pour la validation d'une transaction :

- FAUX n'engendre pas VRAI :  $(\forall i : \neg P_i.v_0) \Rightarrow (\forall i : \neg P_i.v_f)$
- VRAI n'engendre pas FAUX :  $(\forall i : P_i.v_0) \Rightarrow (\forall i : P_i.v_f)$
- Une seule valeur initiale fausse entraîne l'invalidation :

$$(\exists i : \neg P_i.v_0) \Rightarrow (\forall i : \neg P_i.v_f)$$

## 2.2 Les différents contextes de résolution

Le problème du consensus peut être envisagé dans différents contextes de communication :

- le support des échanges d'information entre processus peut être aussi bien une mémoire partagée accessible aux différents processus pour communiquer (architecture multiprocesseurs dotée d'une zone mémoire globale) aussi bien qu'un simple réseau de communication par messages (architecture distribuée) pour communiquer entre processus,
- système synchrone ou système asynchrone : les processus s'exécutent en synchronisme réel ou sont indépendants les uns des autres.

La difficulté survient dès que l'on se place dans un contexte où tous les composants du système ne sont pas fiables et peuvent donc provoquer des fautes. Les défaillances peuvent se situer au niveau des communications (perte de message par exemple) ou au niveau des processus (processeurs). Pour ces derniers, il faudra distinguer d'une part l'arrêt pur et simple d'un processus et d'autre part, le passage à un comportement arbitraire (comportement byzantin).

Un autre paramètre important est, dans le cas d'une communication par messages, l'existence ou non d'une borne connue sur le délai de transfert d'un message. L'existence d'un délai maximum connu permet de retrouver des hypothèses proches de celle d'un système synchrone au sens strict.

Un résultat important et « symbolique » est qu'il n'existe pas d'algorithme résolvant le consensus dans le cas d'un système asynchrone dès qu'un seul processus peut être défaillant (par arrêt).

## 2.3 Preuve de l'impossibilité du consensus en asynchrone

Il est important de bien préciser les hypothèses :

- Chaque processus évolue à son rythme (système asynchrone) ;
- Les processus communiquent par un protocole point-à-point <sup>1</sup> ;
- Il n'y a pas de délai maximal connu pour la transmission des messages ;

---

<sup>1</sup>Même si le protocole est un protocole de diffusion, il n'existe pas de solution si ce protocole n'est pas ordonné

La preuve, due à M. Fisher, N. Lynch et M. Paterson [FLP85], repose sur la mise en évidence d'une séquence d'exécution empêchant tout consensus en présence d'une défaillance d'un **seul** processeur. Plus exactement, l'arrêt d'un processus peut entraîner l'impossibilité de conclure.

De façon standard, l'exécution du protocole peut être abstraite sous la forme des événements engendrés par les messages et en particulier on s'intéresse aux réceptions des messages. On note  $E(t)$  l'état courant du système à l'instant  $t$  composé, classiquement, d'une part par les états locaux des processus et d'autre part, par les messages en transit.

**Définition 2.1** *Un état  $E(t)$  est dit déterministe s'il conduit toujours à la même valeur finale consensuelle. Dans le cas contraire, il est dit bivalent.*

La démonstration repose sur les deux lemmes suivants :

**Lemme 2.1** *Il existe un état initial bivalent.*

**Lemme 2.2** *Partant d'un état bivalent, on peut trouver (il existe potentiellement) une séquence non vide de transitions vers un autre état bivalent.*

L'existence d'un état initial bivalent (lemme 2.1) et l'existence d'un « chemin infini » d'état bivalent à état bivalent amène à conclure à l'inexistence du protocole.

Pour simplifier, on suppose que les processus doivent se mettre d'accord sur une valeur booléenne. Initialement, ils possèdent donc chacun une valeur V ou F. Les processus devront se mettre d'accord sur l'une de ces deux valeurs.

### Preuve du lemme 2.1

Si l'on considère les états initiaux possibles, on peut constater qu'il est possible de les ordonner de telle façon que 2 états adjacents ne diffèrent que par une seule valeur correspondant à un processus fixé.

	$P_1$	$P_2$	...	$P_{N-1}$	$P_N$	Résultat
$E_1$	V	V	...	V	V	V
$E_2$	F	V	...	V	V	?
$E_3$	F	F	...	V	V	?
			...			
$E_N$	F	F	...	F	V	?
$E_{N+1}$	F	F	...	F	F	F

Supposons que tous les états initiaux soient déterministes ( $\equiv$  pas d'état initial bivalent). Il existe alors forcément parmi ces états  $E_i$ , deux états  $E_v$  et  $E_f$  l'un conduisant à la valeur consensuelle  $V$  et l'autre à la valeur  $F$  mais ne différant que par un seul composant :

$$\forall i \neq i_0 : E_v[i] = E_f[i] \wedge E_v[i_0] = \neg E_f[i_0]$$

Ce qui revient à dire qu'il faut donc qu'il existe une fonction *Consensus* prenant en paramètre les états et telle que :

$$\text{Consensus}(E_v) = \neg \text{Consensus}(E_f)$$

Or, si le processus  $i_0$  est à l'arrêt dès le début du protocole (ou avant d'avoir pu communiquer quoi que se soit aux autres processus) les deux états précédents deviennent :

$$\forall i \neq i_0 : E'_v[i] = E'_f[i] \wedge E'_v[i_0] = E'_f[i_0] = \perp \equiv E'_v = E'_f$$

Et par conséquent, on aura  $\text{Consensus}(E'_v) = \text{Consensus}(E'_f)$ . Les deux configurations  $E_v$  et  $E_f$  conduisent donc alors au même consensus contrairement à l'hypothèse faite.

Il existe donc bien des états initiaux bivalents.

## Preuve du lemme 2.2

Partant d'un état bivalent  $E(t)$ , il existerait un algorithme si aucune séquence de transition vers un autre état bivalent ne pouvait être trouvée. Supposons donc que partant un état bivalent, il existe un événement de réception  $r_v$  d'un message provoquant le passage dans un état déterministe conduisant à  $V$ , et il existe donc aussi un événement de réception  $r_f$  d'un message provoquant le passage dans un état déterministe conduisant à  $F$ . L'hypothèse d'un état bivalent impose l'existence de ces deux événements.

**Cas 1 : les deux événements  $r_v$  et  $r_f$  ont lieu dans des processus récepteurs distincts** L'occurrence de ces deux événements peut avoir lieu dans n'importe quel ordre et donne le même état résultat. Or, selon l'ordre d'occurrence, l'état résultat serait  $V$  ou  $F$ . Un même état ne peut donner un résultat différent.

**Cas 2 : les deux événements  $r_v$  et  $r_f$  ont lieu dans le même processus** Dans ce cas, un premier événement peut avoir lieu et le processus peut s'arrêter. Alors, l'état atteint ainsi est le même quel que soit l'événement arrivé avant la panne. Or, l'état avant la panne avait conduit à décider  $V$  si l'événement était  $r_v$  et à décider  $F$  si l'événement était  $r_f$ . On voit donc qu'à nouveau une contradiction existe.

L'hypothèse initiale supposant qu'il n'existait pas de séquence possible de transitions vers un autre état bivalent était donc fausse.

## 2.4 Un algorithme simple

Pour que tous les processus prennent la même décision, évaluent le même résultat final, il suffit qu'ils aient connaissance des toutes les valeurs initiales et qu'ils utilisent une même fonction pour évaluer la valeur finale :

$$\forall i : P_i.v_f = F(P_0.v_0, \dots, P_{N-1}.v_0)$$

Initialement, un processus  $i$  ne connaît que  $P_i.v_0$  et il ne peut donc évaluer  $F$ .

Si les communications sont fiables, un algorithme résolvant le problème peut être très simplement trouvé. Chaque processus diffuse sa valeur initiale à tous les autres processus. Il recevra donc lui-même la valeur initiale de tous les autres processus. Une fois en possession de toutes ces valeurs, chaque processus pourra prendre une décision à partir des **MÊMES** données. Par conséquent, la fonction  $F$  donnera le même résultat quel que soit le processus.

## 2.5 Quelques problèmes voisins

Le problème du consensus comporte de nombreuses variantes applicatives. Parmi elles, on peut en distinguer deux classiques :

- Un processus « Maître »  $m$  propose une valeur  $V_m$  qu'il diffuse aux autres et finalement, tout processus correct choisit la même valeur qui peut être soit la valeur proposée  $V_m$  si le processus « maître » était correct, soit une valeur par défaut si celui-ci était incorrect. Ce problème est plus connu sous le terme de problème des généraux byzantins. Il a été largement étudié dans le domaine de la tolérance aux fautes.
- Chaque processus propose une valeur  $v_i$  et les processus corrects doivent finalement construire un vecteur  $Vd_i$  identique dans lequel  $Vd_i[j] = v_j$  pour tout processus  $i$  correct. C'est de cette façon que des processus peuvent se mettre d'accord sur un état global par exemple.

## 3 Conclusion

L'absence d'algorithme pour obtenir un consensus dans le cas d'un système asynchrone n'empêche pas de trouver des solutions dès que l'on change quelque peu les hypothèses sur les propriétés de la communication. En effet, il suffit, par exemple, d'avoir un délai borné connu pour la transmission des messages, pour obtenir un cadre où des algorithmes existent. L'existence d'hypothèses minimales a même été démontrée. Il faut

introduire la possibilité pour chaque participant d'avoir une certaine connaissance de l'état des autres. Cette connaissance peut être cependant imparfaite. La modélisation consiste à introduire le concept de détecteur de défaillance (failure detector)[CHT96]. Un tel détecteur est associé à chaque processus participant et est capable de détecter la défaillance (l'arrêt) des autres processus. Les propriétés garanties par les détecteurs de défaillance peuvent prendre diverses formes.

À titre d'exemple, les plus faibles propriétés que doivent garantir les détecteurs sont dénotées par  $\diamond\mathcal{W}$  :

**Condition faible  $\diamond\mathcal{W}$**

- Complétude faible : finalement tout processus défaillant est suspecté de façon permanente par au moins un processus correct ;
- Exactitude faible : il existe une date à partir de laquelle au moins un processus correct n'est pas suspecté par tous les autres processus corrects.

Un autre type de détecteur, dénoté  $\diamond\mathcal{S}$ , garantit une complétude plus forte sur la détection des défaillances :

**Condition forte  $\diamond\mathcal{S}$**

- Complétude forte : finalement tout processus défaillant est suspecté de façon permanente par tous les processus corrects ;
- Exactitude faible : il existe une date à partir de laquelle au moins un processus correct n'est pas suspecté par tous les autres processus corrects.

L'existence de ces détecteurs vérifiant les conditions  $\diamond\mathcal{W}$  ou  $\diamond\mathcal{S}$  permet de développer des algorithmes corrects apportant des solutions au problème du consensus dans un contexte restant fondamentalement asynchrone et non fiable [Sch97].

Enfin, le problème du consensus est équivalent au problème d'une diffusion vers un groupe (multicast) fiable totalement ordonnée. En particulier, si l'on dispose d'un tel protocole de diffusion, le problème du consensus peut être simplement résolu. Il suffit que chaque processus diffuse sa valeur au groupe. La première valeur reçue par chaque processus est la même et peut donc constituer la valeur choisie par tous.

## Références

- [CHT96] T.D. Chandra, V Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of ACM*, 43(4) :685–722, 1996.
- [FLP85] M. Fischer, N. Lynch, and M Paterson. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32(2) :374–382, April 1985.
- [Sch97] A. Schiper. Early consensus in an asynchronous system with weak failure detector. *Distributed Computing*, 10(3) :149–157, 1997.