INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

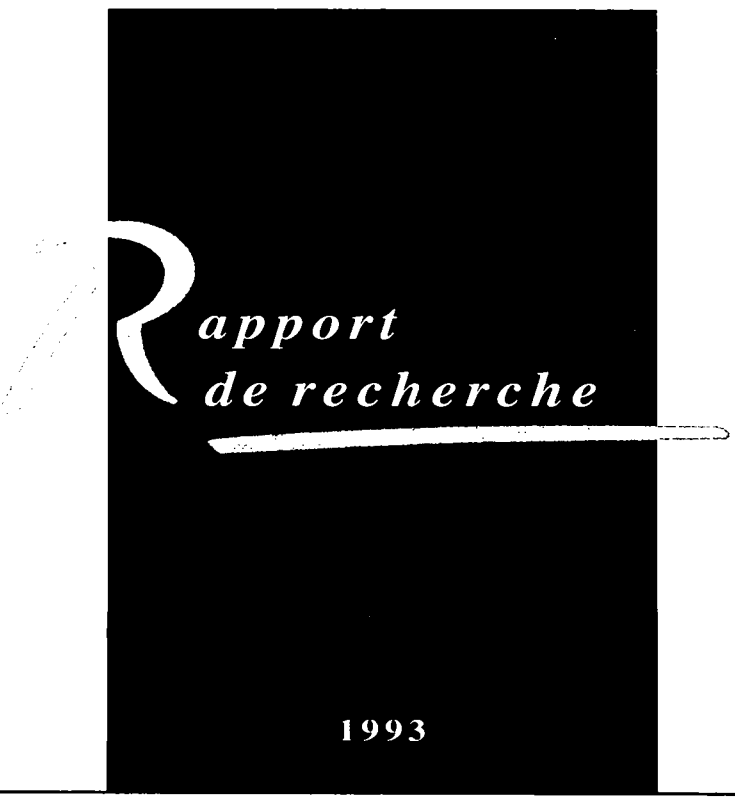
Concepts et problèmes de l'algorithmique répartie

Michel RAYNAL

N° 1842
Janvier 1993

PROGRAMME 1

Architectures parallèles,
Bases de données,
Réseaux et Systèmes distribués



IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Concepts et problèmes de l'algorithmique répartie

Michel RAYNAL
Programme 1, projet ADP (Algorithmes Distribués et Protocles)
Janvier 1993

Publication Interne n° 697 - 18 pages

Résumé

Cet article se veut une introduction informelle à l'algorithmique répartie. Après avoir précisé certaines des caractéristiques fondamentales du contexte réparti, trois paradigmes sont étudiés : l'exclusion mutuelle, l'ordre causal et la détection de la terminaison. Des éléments permettant de mieux comprendre la dynamique des calculs répartis sont ensuite donnés ; on y trouve notamment une description de mécanismes d'horlogerie logique ainsi que la définition et la capture d'états globaux.

Concepts and problems in distributed computing

Abstract

This paper constitutes an informal introduction to some problems of distributed computing. Three paradigms are addressed (mutual exclusion, termination detection and causal order) to exhibit some fundamentals aspects of the distributed setting. Some elements that give a better understanding of distributed computations are also displayed.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE (U.R.A 227) UNIVERSITE DE RENNES I I.N.S.A. DE RENNES
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE (UNITE DE RECHERCHE DE RENNES)

Concepts et problèmes de l'algorithmique répartie*

Michel RAYNAL
IRISA
Campus de Beaulieu
35042 Rennes Cédex
raynal@irisa.fr
fax : 99.38.38.32

Résumé

Cet article se veut une introduction informelle à l'algorithmique répartie. Après avoir précisé certaines des caractéristiques fondamentales du contexte réparti, trois paradigmes sont étudiés : l'exclusion mutuelle, l'ordre causal et la détection de la terminaison. Des éléments permettant de mieux comprendre la dynamique des calculs répartis sont ensuite donnés ; on y trouve notamment une description de mécanismes d'horlogerie logique ainsi que la définition et la capture d'états globaux.

Mots-clés : Algorithme réparti, communication asynchrone, ordre partiel, causalité, exclusion mutuelle, ordre causal, terminaison, temps logique, état global, dynamique des calculs répartis.

1 Introduction

Issue des protocoles et de problèmes auxquels on se trouve confronté dans un contexte de vrai parallélisme et de distribution géographique, l'algorithmique répartie est devenue un des éléments clés dont la connaissance s'avère nécessaire dès que l'on désire maîtriser les applications et les systèmes qui font intervenir plusieurs sites qui ne peuvent s'échanger des informations qu'à l'aide de messages. En effet bien peu du savoir acquis en algorithmique séquentielle ou parallèle (utilisant une mémoire centrale comme lieu d'échange et de coopération) s'avère directement utilisable dans un contexte de répartition. En d'autres termes un tel contexte nécessite, pour résoudre les problèmes qui s'y posent, la connaissance de concepts, d'outils et d'algorithmes propres, une approche empirique s'y révélant très vite limitée [1,17].

Aussi a-t'on vu paraître dans les années 80 de nouvelles revues et conférences et un certain nombre d'ouvrages, tant en France qu'à l'étranger, totalement consacrés aux algorithmes répartis et aux systèmes distribués. Un corpus de connaissances s'est ainsi dégagé et parmi celles-ci, certaines tendent à s'imposer comme constituant des bases dont la maîtrise est nécessaire à qui veut concevoir et implémenter des applications et des systèmes répartis [22].

*Article invité au congrès AFCET 93, VERSAILLES, 8-10 JUIN 1993

Le but de cet article n'est pas de présenter de telles bases mais, à travers quelques définitions, exemples et principes algorithmiques, de permettre une meilleure appréhension de la répartition et de sa problématique. L'article est ainsi divisé en 3 parties principales. La partie 2 caractérise la répartition, du point de vue algorithmique ; la partie 3 aborde 3 paradigmes de l'algorithmique répartie (exclusion mutuelle, communication causale et détection de la terminaison) et permet à ce titre de mieux cerner certaines des spécificités du réparti. Enfin la partie 4 s'intéresse à la dynamique des calculs répartis : quelle est leur nature, quels outils mathématiques peuvent être utilisés pour les analyser, comment détecter et étudier leurs propriétés.

Cette présentation, non exhaustive, se veut informelle. Aussi le lecteur désireux d'approfondir un point particulier pourra consulter les références bibliographiques qui accompagnent l'exposé. En un mot cet article est essentiellement introductif et son esprit est celui d'un tutoriel.

2 Ce qui caractérise le réparti

2.1 Structure d'un algorithme réparti

Un algorithme réparti est formé d'un certain nombre de processus séquentiels qui ne peuvent s'échanger de l'information, ces échanges étant rendus nécessaires pour la réalisation d'un but commun, qu'à l'aide de messages. Il n'y a donc pas de mémoire centrale accessible à l'ensemble des processus. Les messages sont transportés par des canaux de communication et c'est en grande partie les propriétés associées aux canaux qui créent la spécificité de l'algorithmique répartie.

Dans le modèle asynchrone les processus évoluent à des vitesses a priori indépendantes et les délais de transfert des messages, bien que finis, sont arbitraires [23]. Dans le modèle à communication synchrone les délais de transfert sont bornés et cette borne est connue de tous [29]. On se placera dans ce qui suit dans un contexte asynchrone (parce qu'il permet de bien mettre en évidence les problèmes posés par le réparti) exempt de défaillances (afin de rester à un niveau introductif). Les propriétés comportementales des canaux sont donc cruciales : délais arbitraires et livraison, pour chaque canal, des messages dans l'ordre de leur émission (ordre fifo) ou non. Ces propriétés ne font que rendre apparentes, au niveau des algorithmes, les caractéristiques du support et de l'environnement physique sur lesquels ils seront exécutés. Pour accentuer cela on suppose généralement que l'algorithme va s'exécuter sur un ensemble de sites (processeurs) connectés par des lignes de communication et qu'à tout processus (respt. canal) est associé un site (respt. ligne). La structure d'un algorithme réparti (ou de son support) peut donc être représentée par un graphe dont les sommets sont les processus et dont les arêtes sont les canaux. Ce graphe, connexe, peut être quelconque ou exhiber une structure régulière (telle qu'anneau, hypercube, etc).

2.2 Exécution d'un algorithme réparti

L'exécution d'un algorithme réparti (ou calcul réparti) est composée d'un certain nombre d'événements. On distingue généralement 3 types d'événements :

- Un événement interne met en jeu un processus et représente l'exécution d'une séquence d'actions qui ne met pas en jeu d'envoi ou de réception de messages.

- Un événement d'émission met en jeu un processus et un canal ; après un tel événement le message émis est en transit sur le canal.
- Un événement de réception met en jeu un processus et un canal qui contient au moins un message ; une réception provoque la consommation d'un message.

Il est habituel de représenter une exécution par un diagramme "espace-temps", tel que celui de la figure 1. On y voit 3 processus P_1, P_2 et P_3 qui ont exécuté un certain nombre d'événements ; un trait horizontal représente l'évolution d'un processus, alors qu'une flèche depuis un trait vers un autre représente un transfert de message ; e_i^j désigne le $j$ième événement de P_i.

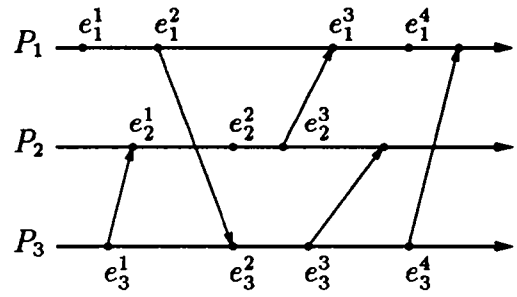


Figure 1 : Une exécution répartie.

Une exécution répartie exhibe des relations de causalité (potentielle) entre événements. Mise en évidence par Lamport en 1978 [16] cette relation exprime le fait que pour tout événement e, certains événements font partie de ses causes : ils définissent son passé causal *PAST(e)*, certains font partie de ses effets : ils définissent son futur causal *FUTURE(e)* et les autres sont indépendants de e : *IND(e)*. La relation de causalité, notée $\rightarrow$ est la suivante :

- si a et b sont 2 événements produits par un même site et si a est produit avant b alors $a \rightarrow b$,
- pour tout message m notant a l'événement émission de m et b l'événement de réception associé on a $a \rightarrow b$,
- enfin si $a \rightarrow b$ et $b \rightarrow c$ alors $a \rightarrow c$.

On a ainsi :

$$\begin{aligned} PAST(e) &= \{a | a \rightarrow e\} \cup \{e\} \\ FUTURE(e) &= \{a | e \rightarrow a\} \cup \{e\} \\ IND(e) &= \{a | \neg(a \rightarrow e) \textbf{ et } \neg(e \rightarrow a) \textbf{ et } a \neq e\} \end{aligned}$$

La relation de causalité permet d'appréhender une exécution répartie comme un ordre partiel sur l'ensemble des événements qu'elle produit. Elle constitue un élément fondamental pour comprendre et analyser la dynamique des calculs répartis. On appelle chemin causal une séquence d'événements $e_0 e_1 e_2 \ldots e_k$ tels que $\forall x \in [1, k]$ on ait $e_{x-1} \rightarrow e_x$.

Du point de vue de la conception d'algorithmes la figure 1 permet d'illustrer une difficulté majeure inhérente au contexte réparti : aucun site (ou processus) ne peut avoir une connaissance instantanée de l'état global dans lequel se trouve le calcul. Si P_i veut obtenir un état global, par réunion de l'état local de chacun des sites il doit demander aux autres leur état local et les messages qui véhiculent ces demandes prennent un temps de transfert arbitraire qui rend impossible une obtention instantanée et qui n'offre aucune garantie quant à la cohérence mutuelle des états locaux obtenus (voir §3.3).

3 Trois paradigmes

Afin d'appréhender des problèmes et des techniques propres à la conception d'algorithmes répartis on présente trois problèmes classiques : l'exclusion mutuelle, la communication causale et la détection de la terminaison. Le premier qui n'est pas propre au réparti va montrer qu'une solution dans un tel contexte, ne consiste pas à "étendre" une solution centralisée. Les deux autres problèmes sont propres au réparti et sont dus à l'arbitraire des délais de transfert et à la difficulté d'obtenir un état global.

3.1 L'exclusion mutuelle

Le problème.
La propriété de sûreté associée à l'exclusion mutuelle indique qu'à tout instant au plus un processus (site) peut être dans la section critique ; la propriété de vivacité spécifie que toute requête d'entrée en section critique doit être satisfaite au bout d'un temps fini [26].

Pour résoudre ce problème deux grandes familles de solutions ont été proposées. Les deux principes algorithmiques distincts sur lesquels elles se fondent (permissions et jetons) peuvent être utilisés pour résoudre d'autres problèmes (notamment de cohérence de données réparties) ; c'est à ce titre que l'exclusion mutuelle constitue un paradigme du contrôle réparti. Le lecteur intéressé par une présentation exhaustive des principes, qui vont être succinctement exposés, pourra consulter [24].

Le principe des permissions.
Les algorithmes de cette famille sont caractérisés par le fait qu'à tout site P_i est associé un ensemble R_i désignant les sites auxquels P_i doit demander la permission ; ce n'est qu'après avoir demandé et obtenu les permissions de tous les sites de R_i que P_i peut entrer en section critique. Selon que la permission qu'a obtenu P_i de P_j doit être restituée à P_j à la sortie de section critique ou non, on distingue les permissions individuelles (ou consommables) et les permissions d'arbitres (ou restituables).

Dans la technique des permissions individuelles, la permission donnée par P_j à P_i a la signification suivante : "en ce qui me (i.e. j) concerne vous (i.e. P_i) pouvez y aller". Le conflit global entre un site P_i et l'ensemble des autres est donc perçu ici, et résolu, comme la somme des *n-1* conflits individuels entre P_i et P_j pour $1 \leq j \neq i \leq n$. Afin de garantir la propriété de sûreté on doit avoir :

$$\forall i \neq j \ : \ j \in R_i \textbf{ ou } i \in R_j \qquad (Spi)$$

Spi garantit que pour tout couple de sites, la requête de l'un pour entrer en section critique, sera connue de l'autre. Il ne peut donc y avoir de requête perçue comme non conflictuelle.

Dans la technique des permissions issues d'arbitres la permission donnée par P_j à P_i a la signification suivante : "en ce qui concerne tous les sites qui me (i.e. P_j) demandent la permission, vous (i.e. P_i) pouvez y aller". Le site P_i devra alors restituer la permission obtenue de P_j après utilisation, afin que P_j puisse satisfaire une autre requête. Afin de garantir la propriété de sûreté on doit avoir :

$$\forall i,j \ : \ R_i \cap R_j \neq \phi \qquad (Spa)$$

Spa garantit que tout couple de sites a au moins un arbitre en commun, qui satisfera donc à un instant donné la requête soit de l'un, soit de l'autre.

Chacune des 2 contraintes *Spi* et *Spa* garantit la sûreté. En ce qui concerne la vivacité une solution consiste à établir un ordre total entre les requêtes pour l'entrée en section critique et à les satisfaire, par le renvoi des permissions demandées, selon cet ordre total. Deux techniques ont été proposées pour construire un tel ordre : l'utilisation d'horloges logiques linéaires [16] ou la gestion d'un graphe acyclique [14].

L'algorithme type fondé sur les permissions individuelles est dû à Ricart et Agrawala [31] ; celui fondé sur les arbitres est dû à Maekawa [18]. Ces deux algorithmes ont donné lieu à de nombreuses variantes [24].

Le principe du jeton.
Le principe est ici extrêmement simple. Afin de garantir la propriété de sûreté on introduit un message spécial, appelé *jeton*, et on astreint tout accès à la section critique à la possession du jeton. L'unicité du jeton assure trivialement la sûreté. La seule chose qu'ont à faire les algorithmes répartis d'exclusion mutuelle consiste donc à réaliser la propriété de vivacité. Pour cela plusieurs solutions sont possibles.

La plus simple consiste à placer les sites sur un anneau et à faire tourner le jeton sur cet anneau ; cet algorithme est appelé "mouvement perpétuel" car personne ne demande le jeton : un site attend que ce dernier lui parvienne. Une autre solution consiste, pour tout site P_i, à demander le jeton lorsqu'il désire pénétrer en section critique. S'il le demande à tous les autres, on parle d'algorithme à diffusion. Le site P_i peut utiliser une heuristique et ne le demander qu'au site P_j auquel il l'avait lui-même transmis après sa dernière utilisation ; ce dernier se chargeant de faire progresser la requête, selon la même heuristique, s'il ne possède pas le jeton. On obtient ainsi des algorithmes fondés sur une structure arborescente, qui évolue dans le temps, et le long de laquelle circulent les requêtes. Les algorithmes les plus connus de cette classe sont [21] et [30]. Le lecteur trouvera un algorithme générique pour cette classe dans [13].

3.2 La communication causale

Définition.
Le caractère arbitraire des délais de transfert (asynchronisme) peut créer dans certains sites une perception erronée de l'ordre dans lequel certains événements se sont produits. La communication causale a pour but d'éliminer cet inconvénient en répercutant sur les réceptions de messages l'ordre causal de leurs émissions respectives, si un tel ordre existe.

De manière plus formelle, "$\rightarrow$" étant la relation de causalité introduite au §2.2, on note *envoi(m)* et $livraison^i(m)$ les événements de diffusion du message m et de la mise à disposition de m à son destinataire P_i. La diffusion causale est telle que (tout site reçoit tout message) :

si $envoi(m_1) \rightarrow envoi(m_2)$
alors $\forall i$: $livraison^i(m_1) \rightarrow livraison^i(m_2)$ **fsi**

Si l'on considère l'exemple de la figure 2 on a donc la livraison de m_3 effectuée après celle de m_1 et de m_2 puisque l'envoi de m_3 suit causalement ceux de m_1 et m_2 ; par contre les envois de m_1 et m_2 étant indépendants, ces 2 messages peuvent être livrés indépendamment l'un de l'autre.

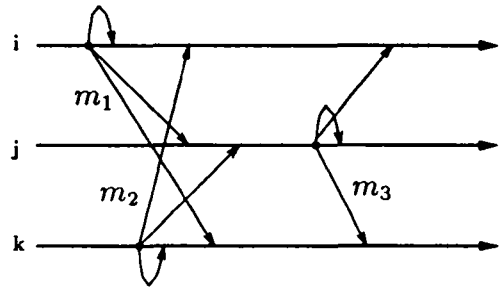


Figure 2 : Diffusions causales.

La diffusion causale a été initialement proposée par Birman et Joseph [2] ; elle a été étendue au cas des communications bi-points dans [27]. De telles primitives, de plus haut niveau que le simple envoi + réception, constituent la base de certains noyaux de système réparti. Leur intérêt réside dans le fait qu'elles cachent l'asynchronisme des supports de communication qui s'avère indésirable ; le programmeur se trouve, grâce à de telles primitives, libéré du "bruit" généré par le support et peut consacrer tout son effort à la seule résolution de son problème.

Une mise en œuvre.
Dans un contexte exempt de défaillances, une mise en œuvre de la communication causale consiste à compter, à l'aide d'un vecteur, le nombre de diffusions effectuées par chaque site et à ne délivrer un message reçu que lorsque les messages, qu'il connaît comme ayant été diffusés avant lui, ont été eux-mêmes délivrés. Pour cela tout message transporte la valeur qu'avait le vecteur de comptage de son émetteur au moment de son émission. $cp_i[k]$ représente le nombre de messages diffusés par P_k, tel que P_i le connaît.

envoi(m) **par** i
$cp_i[i] := cp_i[i] + 1;$
$diffuser(m, cp_i)$

réception(m, cp) **par** i **en provenance de** j
si $i = j$ **alors** $livraison(m)$
sinon attendre $(cp_i[k] \geq cp[k] \ : \forall\ k \neq j)$
et $(cp_i[j] + 1 = cp[j])$;

$livraison(m)$;
$cp_i[j] := cp_i[j] + 1$
fsi

Le lecteur pourra remarquer que si l'on ne considère que 2 sites avec un seul émetteur et un seul récepteur, la communication causale se réduit à l'ordre fifo.

3.3 La détection de la terminaison

Le problème.
Comme le précédent il s'agit là d'un problème typique du calcul réparti qui n'aurait que peu de sens dans un autre contexte. Il s'agit de détecter la fin d'une exécution. La propriété de sûreté est : si l'on annonce la fin du calcul alors celui-ci est bien terminé (cohérence). La propriété de vivacité traduit la dynamique de cette détection : si le calcul se termine, on doit annoncer sa terminaison. Il s'agit comme on le voit d'un problème d'observation de calculs répartis.

La difficulté de sa résolution réside dans l'incapacité, qu'a un algorithme réparti, de capter instantanément un état global cohérent. L'algorithme qui observe subit le même asynchronisme que l'application observée et il est en conséquence nécessaire d'obéir à certaines règles si l'on désire effectuer une observation cohérente.

On considère généralement que l'application obéit aux 2 règles suivantes :

R1 : tout processus actif peut exécuter des instructions, envoyer des messages, ou devenir passif.

R2 : un processus passif ne peut redevenir actif que lorsqu'il reçoit un message.

et l'on dit que l'application est terminée lorsque tous les processus sont actifs et tous les canaux sont vides [20].

Éléments de solution.
Les algorithmes de détection de terminaison répartie sont fondés sur une structure itérative : le concept de *vague* [29,32,34]. Une vague est un flot de contrôle qui visite tous les sites/processus de l'application ; lorsque la vague retourne à son initiateur, celle-ci lui ramène les valeurs confiées par les sites lorsqu'ils ont été visités ; l'initiateur utilise ces valeurs et relance éventuellement une nouvelle vague.

Dans le cas de la terminaison un site visité par une vague en stoppe la progression tant qu'il est actif. Plusieurs solutions sont envisageables selon les informations confiées par le site à la vague. Un algorithme simple et élégant dû à Mattern [20] consiste à faire compter par tout processus i les nombres de messages s_i et r_i qu'il a respectivement émis et reçus depuis le début, et à les confier à la vague lorsqu'elle le visite. Après avoir obtenu toutes les réponses (s_i, r_i) transportées par la vague l'initiateur calcule :

$$S = \sum_i s_i \qquad R = \sum_i r_i$$

et compare la valeur R obtenue par la précédente vague et appelée R' avec S ; si $S = R'$ alors l'application est terminée ; par contre on ne peut rien conclure de $S = R$ comme le montre la

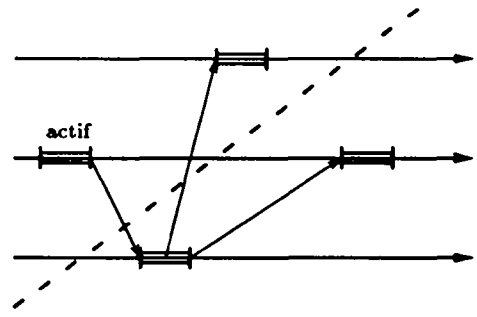


Figure 3 : Une vague visite les sites.

figure 3 dans laquelle le trait en pointillés symbolise la visite de la vague dans chaque site.

Le lecteur intéressé par de tels algorithmes se reportera à [3,9,15,20,24]. Selon la technique choisie pour réaliser les vagues on obtiendra autant d'algorithmes particuliers. Les supports "canoniques" pour les vagues sont les anneaux et les structures arborescentes ; les premiers sont caractérisés par une visite séquentielle alors que les seconds permettent de réaliser une visite parallèle. En ce qui concerne la terminaison, on notera que, dans le cadre asynchrone, les techniques de comptage constituent l'élément de base des solutions.

4 Comprendre les calculs répartis

4.1 Horlogerie logique

Nous avons montré au §2.2 qu'une exécution répartie pouvait être représentée par un ordre partiel sur les événements qu'elle produit, la relation d'ordre y traduisant la notion de causalité entre événements. Il s'agit là d'un élément fondamental pour comprendre la structure et la dynamique des calculs répartis, et en conséquence leurs propriétés.

Afin d'utiliser cette relation d'ordre, que ce soit pour comprendre les calculs répartis ou pour aider à la conception de nouveaux algorithmes, des auteurs ont proposé des systèmes d'horlogerie logique qui permettent de dater les événements, la datation produite respectant la relation de causalité. Chaque site est doté d'une horloge logique qui mesure sa propre progression et d'une représentation du temps logique global avec laquelle il date les événements. Selon la structure de données utilisée pour représenter le temps global on obtient le temps linéaire de Lamport ou le temps vectoriel de Fidge-Mattern. Une synthèse du sujet se trouve dans [28].

Le temps linéaire de Lamport.
Chaque site est pourvu d'une horloge logique h_i, une variable entière à valeurs croissantes, gérée par les règles suivantes :

R1 : avant un événement interne ou une émission, le site P_i effectue $h_i := h_i + 1$.

R2 : tout message transporte sa date logique d'émission (valeur de l'horloge de l'émetteur).

R3 : à la réception d'un message *(m,h)* le site récepteur P_i effectue le recalage d'horloge : $h_i := max(h_i, h) + 1$ avant de dater la réception.

Il est facile de voir que ce temps logique croît le long d'un chemin causal et si un événement e est daté h, il y a h-1 événements sur le plus long chemin causal (séquentiel) qui aboutit à e.

Ce type de datation est très utilisé pour obtenir un ordre total (cf. §3.1). On associe à tout événement une estampille constituée de sa date et de son site d'occurrence. Deux événements a et b, respectivement estampillés (h,i) et (k,j) sont alors ordonnés par l'ordre lexicographique de leurs estampilles :

$$a \textbf{ avant } b \Leftrightarrow (h < k \textbf{ ou } h = k \text{ et } i < j)$$

Ceci suppose que les sites ont des identités distinctes et comparables. L'ordre total obtenu est cohérent avec la causalité. Cette représentation du temps est due à Lamport[16] ; l'horloge qui mesure la progression d'un site P_i et sa représentation du temps global sont ici confondues dans une seule variable h_i.

Le temps vectoriel de Fidge-Mattern.
Chaque site P_i est ici doté d'un vecteur d'entiers $v_i[1..n]$ dans lequel $v_i[i]$ mesure la progression de P_i (en nombre d'événements) ; le vecteur v_i, au complet, représentant le temps global tel qu'il est perçu par le site P_i [10,19].

Les règles de gestion du vecteur v_i sont les suivantes :

R1 : avant tout événement (interne, émission, réception), le site P_i effectue $v_i[i] := v_i[i] + 1$.

R2 : tout message émis par un site P_i transporte sa date logique d'émission v_i.

R3 : à la réception d'un message (m,v) le site P_i effectue (outre R1) le recalage de sa perception du temps global :

$$\forall x \ : \ v_i[x] := max(v_i[x], v[x])$$

En définissant : $vh < vk \Leftrightarrow \forall x \, vh[x] \leq vk[x]$ **et** $\exists x \ : \ vh[x] < vk[x]$ ce système d'horlogerie code exactement l'ordre partiel entre les événements ; en d'autres termes si a et b sont 2 événements datés respectivement vh et vk on a :

$$a \rightarrow b \Leftrightarrow vh < vk$$

Cette technique de datation est très utile dès que l'on désire capter la causalité ; c'est le cas dans nombre d'applications comme la mise au point répartie et le maintien de la cohérence de données réparties.

4.2 État global

Définitions.
Étudier les propriétés d'un calcul nécessite souvent d'en calculer un état et de vérifier ensuite si celui-ci possède ou non telle propriété. Dans le cas des exécutions réparties il est tout d'abord nécessaire de définir ce que l'on entend par état global. Un tel état est constitué d'un état local de chacun des processus et de l'état de chacun des canaux de communication. L'état d'un canal (unidirectionnel) est constitué par l'ensemble (la séquence si le canal est fifo) des messages dont l'émission est captée dans l'état local de l'émetteur et dont la réception ne l'est pas dans celui du récepteur. De plus un état global est dit cohérent si les états locaux des processus ne font état d'aucune réception de messages pour lesquels les émissions ne sont pas enregistrées dans

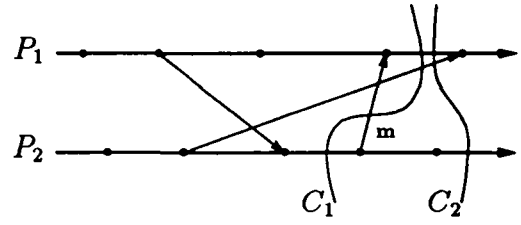


Figure 4 : Coupes et états globaux.

les états locaux de leurs émetteurs [5].

On peut associer à tout état global une coupe. Il s'agit de l'ensemble des événements dont rendent compte les états locaux des processus relativement à un état global. Sur la figure 4 une coupe est indiquée par un trait du haut vers le bas ; C2 correspond à un état global cohérent, alors que C1 est associé à un état incohérent (le message m est reçu pour P_1 et non encore émis pour P_2).

L'algorithme de Chandy-Lamport.
Il existe plusieurs algorithmes de calcul d'un état global cohérent. Le premier d'entre eux est dû à Chandy-Lamport et considère que les canaux sont fifos. Afin de garantir la cohérence deux états de contrôle pour les processus sont utilisés (ils sont symbolisés par des couleurs ; bleu/rouge : l'état local n'a pas été/a été pris) et des messages spéciaux ou marqueurs sont utilisés. Les règles de l'algorithme sont les suivantes [5] :

- initialement tous les processus sont bleus ;
- un processus bleu peut prendre son état local lorsqu'il le désire : il devient alors rouge et envoie des marqueurs sur chacun de ses canaux de sortie ;
- un processus bleu qui reçoit un marqueur doit prendre son état local (il devient alors rouge et envoie des marqueurs) ;
- un processus rouge qui reçoit un marqueur sur un canal d'entrée considère la séquence des messages reçus entre la prise de son état local et la réception de ce marqueur comme constituant l'état du canal.

Il est facile de voir que l'état global est cohérent : en effet un message émis sur un canal par un processus rouge est émis apès un marqueur et ne peut donc être consommé que par un processus rouge.

Cet algorithme est intéressant pour détecter des propriétés stables (telles que la terminaison ou l'interblocage), c'est à dire des propriétés qui, une fois vraies, le restent [12,14,15]. L'état global S rendu en résultat possède en effet la propriété suivante : il est accessible depuis l'état dans lequel se trouvait l'application au moment du lancement de l'algorithme de calcul de S, et l'état dans lequel se trouve l'application lorsque l'on obtient S est accessible depuis S; par contre rien n'indique que le calcul soit réellement passé par S. Cet algorithme ne peut donc être utilisé pour calculer l'occurrence de propriétés instables ; dans ce cas des algorithmes plus sophistiqués (qui captent des états locaux et leur associent des dates vectorielles pour tester

leur indépendance causale) doivent être utilisés [8,11].

La géométrie des calculs répartis.
Dans [7] Charron-Bost a mis en évidence une propriété très intéressante des calculs répartis. Considérons la figure 5 qui donne un exemple de calcul réparti dans lequel à chaque événement est associée sa date vectorielle.

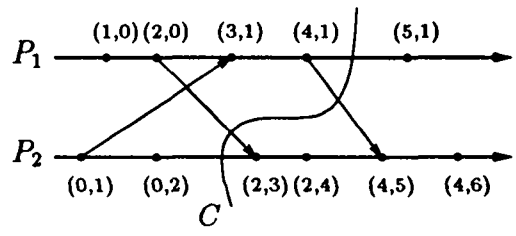


Figure 5 : Une exécution avec estampilles.

La représentation géométrique associée à ce calcul est la suivante. On considère un espace à 2 dimensions (une par processus) et l'on trace les 2 lignes *L1* et *L2* définies par les dates vectorielles des événements produits par chacun des processus P_1 et P_2 (figure 6). La zone comprise entre les lignes *L1* et *L2* (incluses) définit un ensemble de dates vectorielles (représentées par des points sur la figure 6) ; chacune de ces dates correspond à une coupe cohérente, c'est à dire à un état global cohérent possible. Par exemple le point *(4,2)* correspond à la coupe *C*. Vérifier une propriété revient alors à étudier celle-ci sur ces points (ou un de leurs sous-ensembles). Le lecteur pourra consulter avec intérêt [6,33] dans lesquels sont également données des définitions réalistes de l'observation d'un calcul réparti.

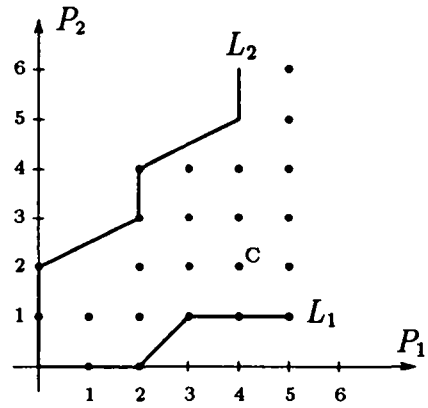


Figure 6 : Représentation géométrique.

5 Conclusion

Cette courte introduction à l'algorithmique répartie n'avait d'autres prétentions que d'en situer le cadre, d'en illustrer quelques problèmes-types et de présenter certains des éléments qui s'avèrent importants lorsque l'on désire analyser les exécutions de ces algorithmes. Le lecteur qui désire en savoir plus pourra consulter les références données tout au long de l'exposé ; celles-ci lui serviront de points d'entrée pour mieux appréhender ce vaste sujet. En aucun cas la bibliographie qui suit ne prétend à l'exhaustivité. Par ailleurs de nombreux sujets liés à l'algorithmique répartie n'ont pu être abordés faute de place ; citons parmi ceux-là, la résistance aux défaillances, la mesure de la concurrence, les mémoires virtuelles réparties, les structures de données réparties, la cohérence de données dupliquées, l'accès aux objets distants, les concepts de groupe et de quorums, la mise au point de programmes répartis, etc. Des éléments de réponses à ces problèmes seront trouvés dans [23,24,25] qui se veut une introduction aux principes algorithmiques des applications et des systèmes répartis.

6 Remerciements

Cette rapide introduction à l'algorithmique répartie serait incomplète si je n'y remerciais pas les personnes dont les discussions m'ont permis de mieux comprendre les algorithmes répartis et, plus particulièrement O. BABAOGLU, J.CL. BERMOND, B. CHARRON-BOST, F. CRISTIAN, D. DOLEV, C. FIDGE, J.M. HÉLARY, PH. INGELS, CL. JARD, G. LE LANN, F. MATTERN, N. PLOUZEAU, A. SCHIPER, M. MIZUNO, F. SCHNEIDER, G. TEL, S. TOUEG, et S. ZACKS.

Bibliographie

[1] BABAOGLU O., MARZULLO K. *Fundamentals concepts in distributed computing.* NATO School, Lisboa, (1992), à paraître, (ACM Press Frontier Series).

[2] BIRMAN K., JOSEPH J. *Reliable communication in presence of failures.* ACM TOCS, vol. 5,1, p. 47-76, (1987).

[3] BRZEZINSKY J., HELARY J.M., RAYNAL M. *Termination detection in a very general distributed computing model.* Rapport de recherche IRISA, (1992), soumis à publication.

[4] CHANDY K.M., MISRA J. *The drinking philosophers problem.* ACM Toplas, vol. 6,4, p. 632-646, (1984).

[5] CHANDY K.M., LAMPORT L. *Distributed snapshots : determining global state of distributed systems.* ACM TOCS, vol. 3,1, p. 63-75, (1985).

[6] CHARRON-BOST B., DELPORTE-GALLET C., FAUCONNIER H. *Local and temporal predicates in distributed systems.* Rapport de recherche LITP 92-36, Inst. B. Pascal, Paris, (1992).

[7] CHARRON-BOST B. *Combinatorics and geometry of consistent cuts : application to concurrency theory.* In Dist. Alg., Springer-Verlag LNCS 392, p. 45-56, (1989).

[8] COOPER R., MARZULLO K. *Consistent detection of global predicates.* Proc. ACM/ONR Workshop on Par. and Dist. Debugging, Santa Cruz, CA, p. 167-174, (may 1991).

[9] DIJKSTRA E.W.D., FEIJEN W.H.T., VAN GASTEREN A.J.M. *Derivation of a termination detection algorithm for distributed computations.* Inf. Proc. Letters, vol. 16,5, p. 217-219, (1983).

[10] FIDGE C.J. *Logical time in distributed computing systems.* Computer, vol. 24,8, p. 28-33,, (1991).

[11] GARG V.J., WALDECKER B. *Detection of unstable predicates in distributed programs.* Tech. report TR 92-07-82, University of Texas at Austin, (1992).

[12] HELARY J.M., JARD CL., PLOUZEAU N., RAYNAL M. *Detection of stable properties in distributed systems.* Proc. 6th ACM PODC, p. 125-136, (1987).

[13] HELARY J.M., MOSTEFAOUI A., RAYNAL M. *A general scheme for token and tree based distributed mutual exclusion algorithms.* Rapport de recherche INRIA 1692, (mai 1992), soumis à publication.

[14] KNAPP E. *Deadlock detection in distributed database.* ACM Computing Surveys, vol. 19,4, p. 303-328, (1987).

[15] LAI T.H., YANG T.H. *On distributed snapshots.* Inf. Proc. Letters, vol. 25, p. 153-158, (1987).

[16] LAMPORT L. *Time, clocks and the ordering of events in a distributed system.* Comm. ACM, vol. 21,7, p. 558-565, (1978).

[17] LAMPORT L., LYNCH N. *Distributed computing : models and methods.* In Handbook of TCS, chap. 18, (Ed. Van Leuwenn), p. 1159-1199, (1990).

[18] MAEKAWA M. *A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems.* ACM TOCS, vol. 3,2, p. 145-159, (may 1985).

[19] MATTERN F. *Virtual time and global states of distributed systems.* Proc. Int. Workshop on Dist. and Par. Alg., North-Holland, p. 215-226, (1987).

[20] MATTERN F. *algorithms for distributed termination detection.* Dist. Computing, vol.2, p. 161-175, (1987).

[21] NAIMI M., TREHEL M. *A distributed algorithm for mutual exclusion based on data structure and fault tolerance.* Proc. IEEE Phœnix Conf., p. 256-276, (1987).

[22] PLOUZEAU N., RAYNAL M. *Elements for a course on the design of distributed algorithms.* ACM Sigcse Bulletin, vol. 24,2, p. 35-40, (1992).

[23] RAYNAL M. *La communication et le temps dans les réseaux et les systèmes répartis.* Eyrolles, Collection DER-EDF, (1991), 230 p.

[24] RAYNAL M. *Synchronisation et état global dans les systèmes répartis.* Eyrolles, Collection DER-EDF, (1992), 230 p.

[25] RAYNAL M. *Gestion des données réparties : problèmes et protocoles.* Eyrolles, Collection DER-EDF, (1992), 200 p.

[26] RAYNAL M. *A simple taxonomy for distributed mutual exclusion algorithms.* ACM Op. Systems Review, vol. 25,1, p. 42-51, (1991).

[27] RAYNAL M., SCHIPER A., TOUEG S. *The causal ordering abstraction and a simple way to implement it.* Inf. Proc. Letters, vol. 39, p. 343-350, (1991).

[28] RAYNAL M. *About logical clocks for distributed systems.* ACM Op. Systems Review vol. 26,1, p. 41-48, (1992).

[29] RAYNAL M., HELARY J.M. *Synchronization and control in distributed systems and programs.* Wiley & sons, (1990), 125 p.

[30] RAYMOND K. *A tree-based distributed algorithm for distributed mutual exclusion* ACM TOCS, vol. 7,1, p. 61-77, (1989).

[31] RICART G., AGRAWALA A.K. *An optimal algorithm for mutual exclusion in computer networks.* Comm. ACM, vol. 24,1, p. 9-17, (1981).

[32] SCHNEIDER F., LAMPORT L. *Paradigms for distributed programs.* In Dist. Systems Springer-Verlag LNCS 190, p. 431-480, (1985).

[33] SCHWARZ R., MATTERN F. *Detecting causal relationship in distributed systems : in search of the holy grail.* Tech. report 215/91, University of Kaiserslautern, (1991).

[34] TEL G. *The structure of distributed algorithms.* Ph. D. Thesis, University of Utrecht (1989), 20 p.

Liste des publications internes Irisa 1993

PI 694 MECANISMES D'ABSTRACTION DANS UNE REPRESENTATION DE LA CONNAISSANCE CENTREE OBJET (R.C.O.)
Stéphane LE PEUTREC, Sophie ROBIN
Janvier 1993, 40 pages.

PI 695 DETECTION DE SEQUENCES ATOMIQUES DE PREDICATS LOCAUX DANS LES EXECUTIONS REPARTIES
Michel HURFIN, Noël PLOUZEAU, Michel RAYNAL
Janvier 1993, 16 pages

PI 696 SEMI-UNIFIED CACHES
Nathalie DRACH, André SEZNEC
Janvier 1993, 18 pages.

PI 697 CONCEPTS ET PROBLEMES DE L'ALGORITHMIQUE REPARTIE
Michel RAYNAL
Janvier 1993, 18 pages.

Unité de Recherche INRIA Rennes
IRISA, Campus Universitaire de Beaulieu 35042 RENNES Cedex (France)

Unité de Recherche INRIA Lorraine Technopôle de Nancy-Brabois - Campus Scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 VILLERS LES NANCY Cedex (France)
Unité de Recherche INRIA Rhône-Alpes 46, avenue Félix Viallet - 38031 GRENOBLE Cedex (France)
Unité de Recherche INRIA Rocquencourt Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)
Unité de Recherche INRIA Sophia Antipolis 2004, route des Lucioles - B.P. 93 - 06902 SOPHIA ANTIPOLIS Cedex (France)

EDITEUR
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

ISSN 0249 - 6399