INRIA

UNITE DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 1044

Programme 3

VERS UNE PROBLEMATIQUE DE L'ALGORITHMIQUE REPARTIE

Jean-Michel HELARY
Michel RAYNAL

Juin 1989

RR_1044

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

VERS UNE PROBLEMATIQUE DE L'ALGORITHMIQUE REPARTIE *

Jean-Michel HELARY, Michel RAYNAL

Publication Interne n° 471 - Mai 1989 - 12 Pages

IRISA, Campus de Beaulieu, 35042 Rennes-Cédex, FRANCE
e-mail : raynal@irisa.fr

TOWARDS A PROBLEMATICS OF DISTRIBUTED COMPUTING

Résumé. Ce papier présente des problèmes que traite l'algorithmique répartie, et une façon de les aborder. Il s'agit d'une présentation informelle et esentiellement introductive du sujet.

Abstract. This paper presents problems adressed by distributed algorithmic and shows how to handle with them. This is an informal and introductive presentation of this subject.

1 Introduction

Le domaine des applications réparties (parfois appelé Informatique Répartie) ne cesse de croitre. Cette avancée de l'utilisation de l'informatique, en tant qu'outil privilégié, dans des domaines de plus en plus divers est essentiellement le résultat du développement de la Science et de la Technique Informatiques. Dans ce développement la conception des algorithmes constitue un domaine fondamental : l'algorithmique a, en effet, pour but, une fois l'existence de solutions établies, de chercher une solution opératoire (la meilleure si possible) pour résoudre un problème donné ; par "opératoire" on entend : définie en termes d'opérations réalisables par une machine. Si l'algorithmique séquentielle a été largement étudiée et présentée, il n'en va pas de même dans un contexte qualifié de parallèle ; dans un tel contexte plusieurs flots de contrôle peuvent coexister simultanément et l'on parle de processus parallèles. Initialement le parallèlisme et son algorithmique ont été abordés et étudiés dans le domaine des systèmes d'exploitation où la conception même du système nécessite une découpe modulaire en activités élémentaires de calcul (les processus) dont il faut gérer les interactions. Le phénomène *réseau* , l'évolution des normes et standard (cf. l'ISO), l'évolution de la technologie (cf. les réseaux de (micro) processeurs) joints aux progrès de la méthodologie de programmation (concepts de type abstrait, paramétrage, processus, port) ont ensuite introduit le concept de communication comme l'un des outils de base dans la conception des algorithmes parallèles ; une algorithmique nouvelle est ainsi apparue : *l'algorithmique distribuée*, dans laquelle les applications ou les systèmes sont considérés comme un ensemble de processus coopérant, à l'aide de messages, à la réalisation d'un but commun (celui auquel le système ou l'application est dédié).

La maitrise d'une conception correcte des systèmes et des applications répartis nécessite en conséquence des concepts, des outils et des algorithmes spécifiques, en un mot ce qu'il est désormais convenu d'appeler l'*algorithmique répartie*.

Un algorithme distribué est ainsi composé d'un ensemble fini de processus séquentiels (s'exécutant sur des machines) connectés par des canaux de communication via lesquels ils s'échangent des messages ; une caractéristique fondamentale est l'absence d'une mémoire centrale qui servirait de lieu d'échanges). Le modèle structurel sous-jacent est donc un graphe fini dont les processus sont les sommets et les canaux les arcs. Les processus peuvent évoluer de façon autonome, aux communications près : un algorithme distribué est donc un algorithme potentiellement parallèle. Si l'on considère un processus, son environnement (formé des processus auxquels il est connecté) est parallèle ; afin d'être sensible au parallèlisme potentiel de son environnement un processus doit être doté d'une structure de contrôle non-déterministe [Hoa84] lui permettant de recevoir, à certains points de contrôle pré-définis, un message d'un de ses voisins

*Papier invité (tutorial), Workshop Franco-Brésilien sur les Systèmes Informatiques Répartis, Florianopolis Brésil (11-14 sept. 1989).

†Ce travail a bénéficié du support du Gréco C3 du CNRS

parmi plusieurs a priori possibles. Les événements qu'un processus peut produire sont : l'émission d'un message, la réception d'un message, une action ne mettant pas de message en jeu ; les 2 premiers événements, à l'opposé du dernier appelé événement local ou interne, mettent en jeu un processus (support du calcul) et un canal (support de la communication).

Les propriétés des processus qui forment un algorithme distribué sont ainsi classiques : un processus est séquentiel (il fait une chose à la fois) et il peut avoir un comportement non- déterministe (à cause du parallèlisme de son environnement). Un soin spécial doit être accordé aux canaux de communication ; il importe en effet de bien définir les propriétés qu'ils sont sensés posséder lorsque l'on définit un algorithme : ces propriétés sont en effet les hypothèses minimales que doit vérifier le support pour que l'algorithme fonctionne. La première de ces propriétés concerne le comportement des canaux : ils peuvent être bornés ou finis (dans le premier cas on connaît un majorant des temps de transit alors que dans le second on sait seulement que ces temps de transit sont finis :les messages arrivent) etc. Un second type de propriété concerne la structure que forment les canaux sur l'ensemble des processus : un anneau, un graphe complet, un arbre, un hypercube, un réseau de DeBruijn, etc, ou un graphe connexe quelconque.

Une autre propriété importante d'un algorithme distribué concerne la *connaissance initiale* que possède chacun des processus. Chaque processus peut ne connaître initialement que les canaux qui les relient à ses voisins (il s'agit là de la connaissance minimale); tous les processus peuvent avoir des identités distinctes et, sans connaître les identités des autres, savoir qu'elles sont toutes distinctes; les processus peuvent savoir ou non le nombre total de processus; ils peuvent ou non connaître la structure globale du réseau dans lequel ils évoluent, etc. Ces hypothèses de connaissance sont importantes à la fois sur les plans théorique et pratique ; théorique car l'existence de solutions à certains problèmes dépend de connaissances minimales qui doivent être satisfaites (il n'existe pas, par exemple, d'algorithme déterministe d'élection dans un anneau anonyme – les processus n'y ont pas d'identités – si le nombre de processus n'est pas premier [Maz87]) ; pratique car si un processus n'a besoin pour participer au calcul local que d'informations locales (il ne connaît par exemple ni le nombre de processus ni la structure du réseau) la modification du nombre de processus ou de la structure n'affectera pas la définition du comportement des processus.

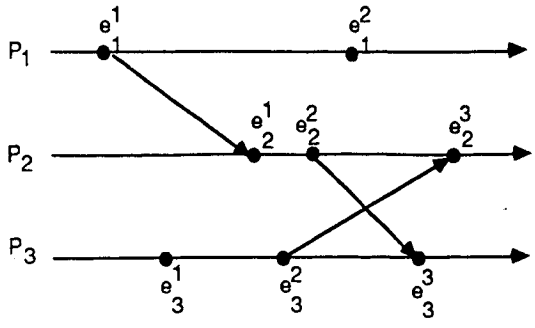

Figure 1 : événements relatifs à une exécution

Un algorithme distribué est donc caractérisé non seulement par la fonction qu'il calcule ou l'invariant qu'il maintient mais aussi par les hypothèses faites sur le comportement et la structure des canaux et la connaissance initiale possédée par les processus (il s'agit là de l'information "statique" qui leur est nécessaire pour participer aux calculs). Ces hypothèses peuvent servir pour comparer différents algorithmes calculant un même résultat ; il s'agit là de critères de comparaison qualitatifs par opposition aux critères quantitatifs que sont les calculs de complexités : nombre de messages, temps de calculs, complexité en bit de la communication.

La spécification d'un algorithme distribué doit donc être constituée de 2 parties : sa signification (le calcul ou le contrôle réalisé) et les caractéristiques de l'environnement dans lequel il est placé.

2 Un problème fondamental

Un des problèmes fondamentaux du calcul distribué est l'impossibilité de capter de façon instantanée l'état global d'une application ou d'un système réparti. Ce problème a été posé par Le Lann en 1977 [Lel77] et une première solution donnée en 1978 par Lamport [Lam78]. Le problème est dû au caractère arbitraire des délais de propagation des messages sur les canaux.

Il est commode de représenter l'évolution d'un système réparti par un diagramme temporel à n dimensions ou un axe horizontal (une dimension) représente (de gauche à droite) l'évolution d'un processus dans le temps (succession d'événements produits par un processus) et où une flèche d'un axe vers un autre représente la communication d'un message (les flèches, également, vont de gauche à droite puisqu'un message ne peut remonter le temps) (figure 1).

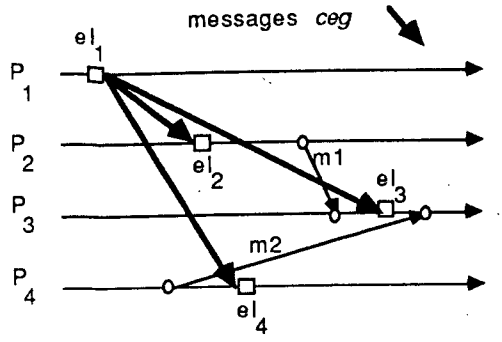

Figure 2 : Un calcul d'état global

L'état global EG d'une application distribuée à l'instant t consiste en un vecteur $(el_1, --, el_n)$ dans lequel el_i est l'état local du processus P_i à l'instant t. Calculer un état global consiste à définir un algorithme qui rend en résultat un tel vecteur, un ou plusieurs processus ayant l'initiative de lancer le calcul de l'état global (on supposera qu'il n'y en a qu'un dans la suite, soit P_1). A l'instant t, P_1 décide donc de calculer un état global : il envoie alors à P_2, P_3 et P_4 un message de contrôle *ceg* ; à la réception de celui-ci, P_i prend une photo de son état et la renvoie à P_1. (figure 2). Le vecteur (el_1, el_2, el_3, el_4) a-t-il un sens ? Si l'on considère le couple (el_3, el_4) : el_4 fait état de l'envoi du message m_2 alors que el_3 ne fait pas état de sa réception ; relativement à ce couple d'états locaux m_2 est en transit. Considérons maintenant le couple (el_2, el_3) ; el_3 fait état de la réception du message m_1 alors que el_2 ne fait pas état de son émission : ces 2 états locaux ne sont donc pas mutuellement cohérents et l'état global résultant n'est pas cohérent : relativement à un état global il peut y avoir des messages émis et non encore reçus mais il ne peut en aucun cas y avoir de messages reçus et non émis [CL85, HPR89]. (Si l'on considère les axes horizontaux comme étant élastiques on peut ramener les états locaux calculés sur la même verticale : l'état global calculé est cohérent si aucun message ne franchit la verticale de la droite vers la gauche).

Les délais arbitraires de propagation des messages constituent ainsi un des problèmes majeurs de l'algorithmique répartie. Afin de pallier cette difficulté 3 solutions ont été proposées. Chacune d'elles offre un schéma d'exécution qui construit une propriété qui permet de concevoir des algorithmes distribués dans un contexte plus intéressant.

La première solution, dûe à Lamport, consiste à utiliser les relations de causalité entre les événements générés par l'application distribuée [Lam78]. Les événements locaux à un processus sont totalement ordonnés, et l'émission d'un message précède dans le temps et causalement sa réception ; l'ensemble des événements produits par l'exécution d'une application répartie forme ainsi un ensemble partiellement ordonné. Lamport a proposé la mise en œuvre d'un temps virtuel global respectant la relation d'ordre partiel sur les événements induite par l'exécution de l'application : à chaque processus est associé une horloge logique, tout message est estampillé avec l'horloge de son émetteur et le récepteur effectue si nécessaire le recalage de son horloge afin que, dans le temps virtuel, tout message soit reçu après avoir été émis. Ce mécanisme de temps virtuel a été utilisé dans de nombreux algorithmes distribués (notamment d'exclusion mutuelle [RA81]) ; il s'avère intéressant chaque fois qu'il est nécessaire d'établir un ordre total, respectant les relations de causalité, sur les événements de l'application (exclusion mutuelle, mise à jour de copies multiples, etc.). Ce principe d'horloge a été étendu par Mattern [Mat89] pour rendre compte des relations d'indépendance entre événements.

La seconde solution consiste à calculer un état global de l'application. (Ce genre de solution est tout à fait adapté aux cas où il faut calculer une propriété sur l'application). Comme on l'a vu le calcul d'un état global cohérent nécessite de synchroniser les prises d'états locaux dans chacun des processus. Le premier algorithme de calcul de tels états a été proposé par Chandy et Lamport en 1985 [CL85], dans le cadre des canaux fifo (c'est-à-dire des canaux sur lesquels l'ordre de réception des messages est identique à leur ordre d'émission) ; il a introduit de plus une algorithmique particulière : l'algorithmique des marqueurs (un marqueur est un message de contrôle spécial qui joue un rôle de "sentinelle" dans un flot de messages séquentiel en séparant les messages émis avant le marqueur de ceux émis après). D'autres algorithmes ont été proposés dans le cadre de canaux non-fifo [LY87, Mat89, Hel89]. Le schéma d'exécution ainsi défini offre une machine virtuelle dans laquelle un processus peut calculer un état global cohérent et donc appliquer sur un tel état un algorithme centralisé ; en un certain sens un algorithme de calcul d'état global construit une abstraction de la mémoire centrale.

La troisième solution consiste à faire une hypothèse simplificatrice sur les temps de transit des messages ; on suppose que le temps de transit et de traitement de tout message est égal à une unité de temps. Les algorithmes conçus sur une telle hypothèse sont appelés algorithmes dis-

tribués synchrones ; ils sont exprimés en termes de pulsations globales qui cadencent l'évolution de l'ensemble des processus et la progression des messages. Cette solution a été proposée par B. Awerbuch en 1985 [Awe85b], qui a donné plusieurs interpréteurs d'algorithmes distribués synchrones sur des réseaux asynchrones ; ils se distinguent par leurs complexités en nombre de messages et en temps et sont appelés synchroniseurs [HR88].

Selon le problème à traiter il peut donc s'avérer intéressant de construire un schéma d'exécution soit qui rende compte d'un temps virtuel global respectant les relations de causalité, soit qui permette de calculer une abstraction de la mémoire centrale, soit qui simule une machine synchrone. Dans tous les cas il s'agit de s'affranchir de l'asynchronisme des délais de communication.

3 Classes d'algortihmes distribués

3.1 Calcul de fonctions

Les algorithmes distribués peuvent être classés en 3 catégories en fonction de leur finalité. On trouve dans la première les algorithmes qui calculent, de façon distribuée, une fonction, c'est à dire donnent un résultat puis s'arrêtent ; la terminaison est ici essentielle. Ces algorithmes font généralement l'hypothèse de canaux de communication fiables, fifo, etc. On y trouve des calculs classiques tels le calcul de chemins de valeurs minimales dans un réseau [Che83, CMH83, HR89], le calcul d'arborescence couvrante de poids minimum [GHS83, LR86, Awe87] et divers calculs dans les graphes [MC82, CM83, KRS84, Awe85a, Cid89].

On y trouve également des calculs de fonctions particulières au contexte distribué tels que le problème de l'élection et le problème de brisure de la symétrie. Dans le premier les processus ont des identités distinctes et la seule connaissance initiale qu'ils ont est celle de leur propre identité et du fait que toutes les identités sont distinctes ; il s'agit de concevoir un algorithme qui peut être lancé par un nombre quelconque de processus (au minimum un seul, au maximum tous) et qui choisit l'un d'entre eux (élection) pour lui faire jouer ensuite un rôle particulier (par exemple choisir un nouveau coordonnateur après une défaillance) ; les processus ayant des identités distinctes et le processus élu étant arbitraire, on choisit généralement celui dont l'identité est un extremum sur l'ensemble des identités. Les premiers algorithmes d'élection proposés supposaient de plus comme connaissance initiale possédée par chaque processus qu'ils étaient placés sur un anneau uni ou bi- directionnel [CR79, HS80, DKM82, Fra82, Pet82, PKR84] ou sur un maillage complet [GM81, KMZ84, AG85] ; des algorithmes sur des topologies quelconques ont ensuite été proposés [HMR87].

Le problème de la brisure de la symétrie consiste également à choisir un processus dans un ensemble mais ici les processus n'ont pas initialement d'identités (ou ce qui revient au même ils peuvent avoir tous la même identité) ; ce problème est beaucoup plus difficile que le précédent : il n'a pas de solution déterministe dans le cas général ; il faut alors introduire des comportements indéterministes (ici au sens de tirage de nombres aléatoires). Il y existe une solution déterministe dans le cas très particulier où les processus sont placés sur un anneau et où le nombre de processus est premier [Maz87].

3.2 Réaliser un service

Une seconde classe d'algorithmes distribués inclut ceux qui réalisent un service que peut utiliser une application ; il s'agit d'algorithmes de contrôle qui implémentent un interpréteur, le maintien d'un invariant ou un protocole de communication. Nous considérons ces 3 cas dans ce qui suit.

3.2.1 Interpréteurs

Les synchroniseurs, dont nous avons parlé au §2, sont un exemple d'interpréteur distribué ; il s'agit d'algorithmes de contrôle dont le but est d'offrir aux utilisateurs, sur une machine asynchrone, une machine virtuelle offrant la communication synchrone (tout message mettant une pulsation pour être transmis et traité). La signification précise de tels interpréteurs est décrite dans [AIR88, Awe85a, HR88].

3.2.2 Maintien d'un invariant

L'illustre représentant de cette classe d'algorithmes de service offerts à des utilisateurs est l'exclusion mutuelle (il s'agit là, à l'heure actuelle d'un des problèmes qui a été le plus étudié en distribué) [Ray84]. Ces algorithmes sont caractérisés par l'invariant qu'ils doivent maintenir (dans le cas de l'exclusion mutuelle, le nombre de processus qui utilisent la section critique est au plus 1) ; on parle souvent à leur sujet d'algorithmes réactifs.

3.2.3 Protocoles de communication

On trouve dans cette classe de service des algorithmes dont le but est de mettre en œuvre des communications ayant certaines propriétés comportementales ; nous en citons 3 : comportement synchrone, comportement fiable et comportement soumis à des contraintes de temps.

Offrir la communication par rendez-vous (type CSP) sur un réseau asynchrone nécessite de mettre en œuvre, à l'aide d'un protocole, des règles de synchronisation qui assurent que le premier processus arrivé au point de communication attendra son interlocuteur (indépendamment du sens de la communication). Un certain nombre de tels protocoles ont été proposés (détaillés dans [Ray85]) ; ils se distinguent par les contraintes plus ou moins fortes qu'ils rajoutent sur le comportement des processus. Il est important de bien noter la différence entre ce type de communication synchrone et celle dont rend compte un synchroniseur : d'une part il y a ici rendez-vous et non le concept de pulsation et d'autre part ce synchronisme ne met en jeu que les 2 processus communicants, à la différence des synchroniseurs qui synchronisent l'ensemble des processus (ils sont tous logiquement à la même pulsation en même temps).

Un grand nombre de protocoles ont été proposés afin de réaliser des communications fiables ; ces algorithmes distribués étendent les protocoles introduits initialement dans les réseaux (cf. les protocoles du type bit alterné qui construisent un médium virtuel fiable sur une ligne non fiable). Le représentant le plus illustre de cette classe est le problème des généraux byzantins [LSP82]. (Une synthèse sur ce sujet est parue dans la version en anglais de [Ray85]). Dans un ensemble de processeurs dont certains ont des comportements aberrants (mais on ne sait pas lesquels) il s'agit pour l'un des processeurs, soit P_0, de diffuser un message à l'ensemble des autres de telle façon que :

- tous les processeurs corrects recoivent la même valeur
- et que cette valeur soit celle que P_0 a émis si P_0 est correct

Il s'agit là d'un problème difficile (considérer le cas où P_0 est incorrect et envoie des valeurs différentes aux autres). Ce problème n'a de solution, d'une part que dans un cadre synchrone [FLP85] et d'autre part que si le nombre de processeurs incorrects est inférieur au tiers du nombre total de processeurs.

Le troisième exemple de protocole implémentant un algorithme distribué de service est celui de la mise en œuvre de communications respectant des contraintes de temps. On trouve ici les algorithmes distribués de recalage d'horloges physiques [LM85, MO83, ST87] qui construisent un temps physique global dans un système réparti et les algorithmes réalisant des communications dites temps-réel [LD87].

3.3 Réaliser une observation

A côté des algorithmes de contrôle réalisant un service, une autre classe d'algorithmes de contrôle est apparue très nettement en distribué : les algorithmes qui réalisent une observation. On retrouve ici à la fois des problèmes connus et de nouveaux problèmes, l'ensemble étant qualifié de problème de recherche de propriétés (stables dans la plupart des cas). Le problème posé par la mise en œuvre d'une observation consiste d'une part à ne pas altérer le comportement de l'application observée et d'autre part à faire une observation cohérente (cf. §2).

La détection de l'interblocage (qu'il soit dû à l'utilisation de ressources communes ou à la communication [CMH83] consiste à observer un circuit dans un graphe des attentes distribué ; il s'agit là d'un problème ancien dans un nouveau contexte. Le problème de la terminaison (un des plus étudiés en distribué) est "apparu" avec le distribué : il s'agit de déterminer si une application distribuée a terminé son calcul. Pour cela on doit observer tous les processus passifs et tous les canaux vides de messages. Le problème introduit par Francez en 1980 [Fra80] constitue le paradigme de l'observation répartie (plus de 60 solutions ont à ce jour été proposées !). Ces 2 problèmes appartiennent à la classe des problèmes de recherche des propriétés stables ; une telle propriété est caractérisée par le fait que dès qu'elle est vérifiée, elle reste vraie. Des algorithmes de calculs de propriété stable ont été proposés (indépendamment d'une propriété de stabilité particulière) ; il s'agit en quelque sorte d'algorithmes distribués génériques [CM86, IIJPR87].

4 Conclusion : Axes de recherche

Pour conclure nous donnons deux axes de recherche qui nous paraissent intéressants (il ne s'agit pas, bien sûr d'une liste exhaustive).

4.1 Méthodologie

Un grand nombre d'algorithmes distribués ont été proposés, souvent *ex abrupto*. Il s'agit maintenant d'organiser cette connaissance du distribué, d'en extraire les concepts et les outils fondamentaux comme cela a été fait pour le séquentiel. Ceci nécessite un effort méthodologique très important [Ray85, Ray87, HR88] ; il faut noter à ce titre l'ouvrage de Chandy et Misra [CM88] qui propose une technique de dérivation qui peut s'avérer prometteuse.

4.2 Champs d'expérimentation

L'existence de machines distribuées (des calculateurs du type hypercube jusqu'aux réseaux locaux par exemple) rend possible l'expérimentation de techniques algorithmiques issues de l'algorithmique répartie. Il importe maintenant de les mettre en œuvre, plusieurs champs d'expérimentation sont possibles : les réseaux locaux temps-réel, et la simulation distribuée en sont 2 exemples parmi beaucoup d'autres. Ce dernier nous intéresse particulièrement car d'une part il présente la plupart des problèmes fondamentaux du réparti [Mis86, IR89], d'autre part les techniques de mises en œuvre possibles sont extrêmement variées [Jef85, FR87] et peuvent donc s'avérer riches d'enseignement, et enfin la simulation séquentielle étant trop coûteuse en temps il est nécessaire d'en explorer d' éventuelles solutions parallèles.

5 Bibliographie

On trouvera dans cette bibliographie une liste de références (non exhaustive), dont celles citées dans le texte. Elles sont rangées par thèmes.

5.1 Livres

[CM88] K. M. Chandy and J. Misra. *Parallel program design : a foundation*. Addison-Wesley, 1988.

[FR87] R.M. Fujimoto and D.A. Reed. *Multicomputer networks: message based parallel processing*. The MIT Press, 1987. 380 p.

[Hoa84] C. A. R. Hoare. *Communicating Sequential Processes*. Prenctice-Hall, 1984.

[HR88] J.-M. Hélary and M. Raynal. *Synchronisation et contrôle des systèmes et des programmes répartis*. Eyrolles, Septembre 1988. English translation to appear, Wiley, 1990.

[Ray84] M. Raynal. *Algorithmique du parallélisme : le problème de l'exclusion mutuelle*. Dunod, 1984. *Algorithms for Mutual Exclusion*. The MIT Press, 1986. 107 p.

[Ray85] M. Raynal. *Algorithmes distribués et Protocoles*. Eyrolles, Septembre 1985. 141 p. *Distributed Algorithms and Protocols*. Wiley, 1988. 161 p.

[Ray87] M. Raynal. *Systèmes répartis et réseaux : concepts, outils et algorithmes*. Eyrolles, Février 1987. *Distributed Computations and Networks*. The MIT Press, 1988. 165 p.

5.2 Calculs de fonctions

[AG85] Y. Afek and E. Gafni. Time and message bounds for election in synchronous and asynghronous complete networks. In *Proc. of 4rd ACM PODC Conference, Minacki, Ontario*, august 1985.

[Awe87] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems. In *Proc. of the 19th Annual ACM Conf. on Theory of Computing*, pages 230–240, New-York city, may 1987.

[Cha82] E. Chang. Echo algorithms: depth parallel operations on general graphs. *IEEE Transactions on Software Engineering*, SE-8(4):391–401, July 1982.

[Che83] A. T. Cheung. Graph traversal techniques and the maximum flow problem in distributed computing. *IEEE Trans. on SE*, SE-9(4):504–512, 1983.

[CM83] K. M. Chandy and J. Misra. Distributed computation on graphs : shortest path algorithm. *Comm. ACM*, 25(11):833–837, November 1983.

[CR79] E.J Chang and R. Roberts. An improved algorithm for decentralised extreme-fiding in circular configuration of processors. *Comm. ACM*, Vol. 22(5):281–283, May 1979.

[Cid89] I. Cidon. An efficient distributed knot detection algorithm. *IEEE trans. on soft. eng.*, SE-15(5):644-649, may 1989.

[DKM82] D. Dolev, M. Klawe, and Rodeh. M. An O(n logn) unidirectional distributed algorithm for extrema finding in a circle. *Journal of Algorithms*, Vol. 3:245–260, 1982.

[Fra82] W. R. Franklin. On an improved algorithm for decentralized extrema finding in a circular configuration of processors. *Comm. ACM*, 25(5):336–337, May 1982.

[GHS83] R. G Gallager, P.A Humblet, and P.M. Spira. A distributed algorithm for minimum weight spanning trees. *ACM Toplas*, 5(1):66–87, Jan. 1983.

[GM81] H. Garcia Molina. Elections in a distributed computing system. *IEE Trans. on Computers*, C 31(1):48–59, january 1981.

[HMPR87] J. M. Hélary, A. Maddi, N. Plouzeau, and M. Raynal. Parcours et apprentissage dans une réseau de processus communicants. *Technique et Science Informatiques*, 6(2):127–139, 1987.

[HMR87] J.M. Hélary, A. Maddi, and M. Raynal. Calcul distribué d'un extremum et du routage associé dans un réseau quelconque. *Inf. Théor. et Appl./Theror. Inf. and Appl.*, 21(3):223–244, 1987.

[HR89] J.-M. Hélary and M. Raynal. Un schéma abstrait d'itération répartie; application au calcul des chemins de valeur minimale. *Techniques et Sciences Informatiques*, 7(3), Mai 1989.

[HS80] D. S. Hirschberg and J. B. Sinclair. Decentralized extrema finding in circular configurations of processors. *Comm. ACM*, 23(11):627–628, november 1980.

[Hum83] P. A. Humblet. A distributed algorithm for minimum-weight directed spanning tree. *IEEE Trans. on Soft. Eng.*, SE 31(6):756–762, july 1983.

[KMZ84] E. Korach, S. Moran, and S. Zaks. Tight lower and upper bounds for some distributed algorithms for complete networks of processors. In *Proc. of 3^d ACM PODC Conference, Vancouver, BC*, august 1984.

[KRS84] E. Korach, D. Rotem, and N. Santoro. Distributed algorithms for finding centers and medians in networks. *ACM Toplas*, 6(3):380–401, july 1984.

[LMT87] K.B Laskmanan, N. Meenakshi, and K. Thulisaraman. A time-optimal message-efficient distributed algorithm for depth first search. *Inf. Proc. Letters*, 25:103–109, 1987.

[LR86] Y. Lavallee and G. Roucairol. A fully distributed minimal spanning tree algorithm. *Inf. Proc. Letters*, 23:55–62, 1986.

[Maz87] A. Mazurkiewicz. Solvability of the asynchronous ranking problem. *Inf. Proc. Letters*, 28:221–224, 1987.

[MC82] J. Misra and K. M. Chandy. A distributed graph algorithm : knot detection. *ACM Toplas*, 4(4):678–686, october 1982.

[MS79] Ph. Merlin and A. Segall. A failsafe distributed routing protocol. *IEEE Trans. on Comm.*, C 27(9):1286–1287, sept. 1979.

[Pet82] J. L. Peterson. An O(n logn) unidirectionnal algorithm for the circular extrema finding problem. *ACM Toplas*, 4(4):758–762, Oct. 1982.

[PKR84] J. A. Pachl, E. Korach, and D. Rotem. Lower bounds for distributed maximum finding. *Journal of the ACM*, 31(4):905–918, october 1984.

[RSS85] D. Rotem, N. Santoro, and J. Sidney. Distributed sorting. *IEEE Trans. on Computers*, C 34(4):372–376, April 1985.

[Seg82] A. Segall. Decentralized maximum flow protocols. *Networks*, 12:213–230, 1982.

5.3 Synchroniseurs

[AIR88] M. Adam, Ph. Ingels, and M. Raynal. The meaning of synchronous distributed algorithms run on asynchronous distributed systems. In *The Third International Symposium on Computer and Information Sciences, Izmir*, November 1988, pp. 307-316.

[Awe85a] B. Awerbuch. Complexity of network synchronization. *Journal of ACM*, 32(4):801–823, October 1985.

[Awe85b] B. Awerbuch. Reducing complexities of the distributed max-flow and breadth-first algorithm by means of network synchronization. *Networks*, 15:425–437, 1985.

[CCGZ87] C.T. Chou, I. Cidon, I.S. Gopal, and S. Zaks. Synchronizing asynchronous bounded delay networks. In *Proc. 2nd. Int. Workshop on Distributed Algorithms*, Amsterdam, July 1987. Springer Verlag LNCS 312 (1988).

[FLS87] A. Fekete, N. Lynch, and L. Shrira. A modular proof of correctness for network synchroniser. In *Proc. 2d Int. Workshop on Distributed Algorithms*, Amsterdam, July 1987. Springer Verlag LNCS 312 (1988).

[LT87] B. Laskmanan and K. Thulasiraman. On the use of synchronisers for asynchronous communication networks. In *Proc. 2d Int. Workshop on Distributed Algorithms*, Amsterdam, July 1987. Springer Verlag LNCS 312 (1988).

[LTC89] B. Laskmanan, K. Thulasiraman and M.A. Comeau. An efficient distributed protocol for finding shortest paths in networks with negative weights. *IEEE Trans. on Soft. Eng.*, SE-15(5):639-644, may 1989.

[PU87] D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. In *6th Annual ACM Symposium on Principles of Distributed Computing*, pages 77–85, August 1987.

[Wel87] J. L. Welch. Simulating synchronous processors. *Information and Computation*, 74:159–171, 1987.

5.4 Terminaison et stabilité

[DFv83] E. W.D. Dijkstra, W. H. D. Feijen, and A. J. M. van Gasteren. Derivation of a termination detection algorithm for distributed computations. *Information Processing Letters*, 16:217–219, 1983.

[DS80] E. W. D. Dijkstra and C. S. Scholten. Termination detection for diffusing computation. *Information Processing Letters*, 11:217–219, August 1980.

[Fra80] N. Francez. Distributed termination. *ACM TOPLAS*, 2(1):42–55, January 1980.

[HJPR87] J.-M. Hélary, C. Jard, N. Plouzeau, and M. Raynal. Detection of stable properties in distributed applications. 6^{th} *ACM SIGACT-SIGOPS, Symp. Principles of Distributed Computing, Vancouver, Canada*, 125–136, August 1987.

[Mat87a] F. Mattern. Algorithms for distributed termination detection. *Distributed Computing*, 2:161–175, 1987.

[Mat87b] F. Mattern. Experience with a new distributed termination detection algorithm. In *Proc. of* 2^{nd} *Int. Workshop on Distributed Algorithms, Amsterdam*, 1987. Springer Verlag LNCS 312 (1988).

[Mis83] J. Misra. Detecting termination of distributed computations using markers. In *Proc. ACM Symposium on PODC, Montréal*, pages 290–294, August 1983.

[Nat84] N. Natarajan. A distributed scheme for detecting communication deadlocks. *IEEE Transactions on Software Engineering.*, SE-12(4):531–537, April 1984.

[SF86] N. Shavit and N. Francez. A new approach to detection of locally indicative stability. In 13^{th} *ICALP, LNCS #226*, pages 344–358, Springer Verlag, 1986.

[Tel86] G. Tel. *Distributed Infimum Approximation.* Tech. report RUU-CS-86-12, University of Utrecht, 1986.

5.5 Horloges logiques, état global, exclusion mutuelle

[CL85] K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM TOCS*, 63–75, February 1985.

[CM86] K. M. Chandy and J. Misra. An example of stepwise refinement of distributed programs: quiescence detection. *ACM TOPLAS*, 8(3), July 1986.

[CMH83] K. M. Chandy, J. Misra, and L. Haas. Distributed deadlock detection. *ACM TOCS*, 1(2):144–156, May 1983.

[Hel89] J. M. Hélary. *Observing global states of asynchronous distributed applications.* Technical Report 468, IRISA, University of Rennes, april 1989. 24 p.

[HPR88] J.-M. Hélary, N. Plouzeau, and M. Raynal. A distributed algorithm for mutual exclusion in an arbitrary network. *The Computer Journal*, 31(4):289–295, 1988.

[HPR89] J.M. Hélary, N. Plouzeau, and M. Raynal. A characterization of a particular class of distributed snapshots. In *Proc. International Conference on Computing and Information (ICCI'89), Toronto*, may 23–27 1989. North Holand.

[LY87] T.H. Lai and T.H Yang. On distributed snapshots. *Inf. Proc. Letters*, 25:153–158, 1987.

[Lam78] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications. of the ACM*, 21(7):558–565, July 1978.

[Lel77] G. Lelann. Distributed systems: towards a formal approach. In *IFIP Congress*, pages 155–160, Totonto, August 1977.

[Mae85] M. Maekawa. A $o(\sqrt{n})$ algorithm for mutual exclusion in decentralized systems. *ACM TOCS*, 3(2):145–159, May 1985.

[Mar85] A. J. Martin. Distributed mutual exclusion on a ring of processes. *Science of Computer Programming*, 5:265–276, 1985.

[Mat89] F. Mattern. Virtual time and global states of distributed systems. In Cosnard, Quinton, Raynal, and Robert, editors, *Proc. Int. Workshop on Parallel and Distributed Algorithms, Bonas, France, oct. 1988*, North Holland, 1989.

[NT87] G. Neiger and S. Toueg. Substituting for real time and common knowledge in distributed sytems. In *6th Annual ACM Symposium on Principles of Distributed Computing*, pages 281–293, August 1987.

[RA81] G. Ricart and A. K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *Comm. ACM*, 24(1):9–17, January 1981.

5.6 Services, protocoles, communication et fiabilité

[BD85] O. Babaoglu and R. Drummond. Streets of byzantium :networks architectures for fast reliable broadcasts. *IEEE Trans. on Soft. Eng.*, SE 11(6):546–554, June 1985.

[Ber80] A. J. Bernstein. Outputs guards and non-determinism in csp. *ACM Toplas*, 2(2):234–238, April 1980.

[BS83] G. N. Buckley and A. Silberschatz. An effective implementation for the generalized input-output construct of csp. *ACM Toplas*, 5(2):223–235, April 1983.

[BSW69] K. A. Bartlett, R. A. Scantlebury, and P. T. Wikinson. A note on reliable full-duplex transmission over half-duplex links. *Comm. ACM*, 12(5):260–265, May 1969.

[CASD85] F. Cristian, H. Aghili, H .R. Strong, and D. Dolev. Atomic broadcast : from simple message diffusion to byzantine agreement. In *Proc. 15th Int. Symposium on Fault-Tolerance Computing*, pages 200–206, 1985.

[Dol82] D. Dolev. The byzantine generals strike again. *Journal of Algorithms*, 3:14–30, 1982.

[DS83] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computers*, 12(4):656–666, 1983.

[FLP85] M. J. Fischer, N. A. Lynch, and M. D. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32(2):374-382, April 1985.

[IGS84] F. B. Ider, D. Gries, and R. D. Schlichting. Fault-broadcast. *Science of Computer Programming*, 4:1–15, 1984.

[LA86] T.V. Lakshman and A. K Agrawala. Efficient decentralized consensus protocols. *IEEE Trans. on Soft. Eng.*, SE 12(5):600–607, 1986.

[Lam84] L. Lamport. Using time instead of timeout for fault-tolerant distributed systems. *ACM Toplas*, 6(2):254–280, April 1984.

[LD87] I. Lee and S. B. Davidson. Adding time to synchronous process communication. *IEEE Trans. on Computers*, C36(8):941–948, 1987.

[LM85] L. Lamport and P.M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, Juanary 1985.

[LSP82] L. Lamport, R. Shostack, and M. Peace. The byzantine general problems. *ACM Toplas*, 4(3):382–401, 1982.

[MO83] K. Marzullo and S. Owiki. Maintaining time in a distributed system. In *ACM Operating Systems Rev.*, pages 44–54, 1983.

[PT86] K. J. Perry and S. Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Trans. on Soft. Eng.*, SE 12(3):477-481, 1986.

[Sch79] A. Silberschatz. Communication and synchronization in distributed systems. *IEEE Trans. on Soft. Eng.*, SE5(6):542–546, 1979.

[Sch82] F. B. Schneider. Synchronization in distributed programs. *ACM Toplas*, 4(2):125–148, 1982.

[Sch86] F.B. Schneider. A paradigm for reliable clock synchronization. In *Proc. Advanced Seminar Real Time Local Area Network*, pages 85–104, April 1986.

[Seg83] A. Segall. Distributed networks protocols. *IEEE Transactions on Inf. Theory*, IT29(1):23–35, January 1983.

[SLL88] B. Simons, J. Lundelius, and N. Lynch. *An Overview of Clock Synchronization.* Technical Report, IBM Research Division, October 1988.

[ST87] T.K. Srikanth and S. Toueg. Optimal clock synchronization. *Journal of ACM*, 34(3):626–645, 1987.

5.7 Simulation distribuée

[CS89] K. M. Chandy and R. Sherman. Space-time and simulation. In *Proc. ECM'89 (Dist. Simulation Conf.), Tampa, Florida*, May 1989.

[Fuj87] R.M. Fujimoto. *Performance measurement of distributed simulations strategies.* Tech. Report UUCS-87-026a, University of Utah, November 1987.

[IR89] Ph. Ingels, M. Raynal. *Simulation répartie de systèmes à événements discrets. Partie 1 : modélisation et algorithmes.* Rapport de Recherche, IRISA, Université de Rennes, juillet 1989.

[Jef85] D.R. Jefferson. Virtual time. *ACM Toplas, Vol. 7, No 3*, 404–425, July 1985.

[Mis86] J. Misra. Distributed discrete-event simulation. In *Computing Surveys, Vol. 18, No 1*, pages 39–65, March 1986.

[MMR88] A.D. Malony, B.D. McCredie, and D.A. Reed. Parallel discrete event simulation using shared memory. *IEEE Trans. on Soft. Eng., Vol. 14, No 4*, 541–553, April 1988.

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 464 **VERS UN MODELE D'ECLAIREMENT REALISTE**
Pierre TELLIER, Kadi BOUATOUCH
66 Pages, Avril 1989.

PI 465 **COMPILATION OF FUNCTIONAL LANGUAGES BY PROGRAM TRANSFORMATION**
Pascal FRADET, Daniel LE METAYER
28 Pages, Avril 1989.

PI 466 **MODEL BASED DIAGNOSIS : A CASE STUDY IN VIBRATION MECHANICS**
Michèle BASSEVILLE, Albert BENVENISTE
26 Pages, Avril 1989.

PI 467 **DELAI DE COMMUNICATION ENTRE NOEUDS VOISINS SUR L'iPSC/2**
Patrice BURGEVIN, André COUVERT, René PEDRONO
16 Pages, Avril 1989.

PI 468 **OBSERVING GLOBAL STATES OF ASYNCHRONOUS DISTRIBUTED APPLICATIONS**
Jean-Michel HELARY
24 Pages, Avril 1989.

PI 469 **ON-LINE MODEL CHECKING FOR FINITE LINEAR TEMPORAL LOGIC SPECIFICATIONS**
Claude JARD, Thierry JERON
16 Pages, Mai 1989.

PI 470 **PROGRAMMING WITH MALI - THE INTERPRETATION OF PROLOG PROGRAMS**
Louis CHEVALLIER, Serge LE HUITOUZE, Olivier RIDOUX
16 Pages, Mai 1989.

PI 471 **VERS UNE PROBLEMATIQUE DE L'ALGORITHMIQUE REPARTIE**
JeanMichel HELARY, Michel RAYNAL
12 Pages, Mai 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique