Un algorithme distribué d'affectation d'identités distinctes aux sites d'un système réparti anonyme

Jean-Michel Hélary, Michel Raynal

INRIA

UNITÉ 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° 806

UN ALGORITHME DISTRIBUE D'AFFECTATION D'IDENTITES DISTINCTES AUX SITES D'UN SYSTEME REPARTI ANONYME

Jean-Michel HELARY
Michel RAYNAL

MARS 1988

RR806

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

Publication Interne n°391
Février 1988
12 Pages

Un Algorithme Distribué d'Affectation d'Identités Distinctes aux Sites d'un Système Réparti Anonyme

Jean-Michel HELARY, Michel RAYNAL

Résumé

Il est généralement fait, lors de la conception d'applications ou de systèmes répartis, l'hypothèse implicite que tous les sites ont des identités distinctes (et ordonnées). Il est alors possible de définir des fonctions d'adressage réparti (algorithmes de routage), et des algorithmes de contrôle qui utilisent les identités des sites lors de prises de décision (exclusion mutuelle, détection ou prévention de l'interblocage, contrôle de la concurrence, etc.). L'attribution d'identités est généralement réalisée statiquement (lors de la définition même des sites). On présente un algorithme distribué d'attribution d'identités distinctes et ordonnées aux sites d'un système réparti connexe, initialement anonyme ; un tel algorithme rend la définition des sites indépendante de valeurs initiales particulières (du point de vue des identités). En plus du résultat apporté, l'algorithme est intéressant dans sa construction méthodique.

Assingning distinct identities to the sites of an anonymous distributed system

Abstract

An implicit hypotheses is usually assumed when designing a distributed system : all sites have distinct (and ordered) identities. It is then possible to define distributed addressing fonctions (eg. routing algorithms), and control algorithms which use sites identities to take decisions (eg. mutual exclusion, deadlock detection or prevention, concurrency control, etc.). Assignment of distinct identities is usually made statically, when sites are defined. A distributed algorithm wich assigns distinct and ordered identities to the sites of a connected anonymous distributed system is presented ; such an algorithm allows the definition of sites to be independent of particular initial values (as far as identities are concerned). Moreover the proposed algorithm is interesting by its methodological construction.

IRISA – Campus de Beaulieu – F-35042 Rennes cédex

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

1 INTRODUCTION.

Un grand nombre d'algorithmes distribués réalisant des fonctions de contrôle dans les applications ou les systèmes répartis, utilisent les identités des sites du système lors de prise de décision [R 87]. C'est par exemple le cas des algorithmes d'exclusion mutuelle fondés sur la technique d'estampillage [L 78, RA 81]. Chaque requête est pourvue d'une identité formée d'un couple *(h, i)* où *h* désigne la valeur qu'avait l'horloge logique [L 78] locale du site émetteur au moment de son émission et où *i* désigne l'identité de ce site. L'affectation d'identités distinctes et ordonnées aux sites, jointe au mécanisme de gestion des horloges logiques (à la Lamport) assure alors que chaque requête posséde une identité distincte et que l'ensemble des requêtes est totalement ordonné. Dans le cas de systèmes synchrones possédant une base de temps physique commune [NT 87], les requêtes peuvent également être pourvue d'une telle identité (*h* est alors une date physique) et l'ensemble des requêtes peut aussi être totalement ordonné. L'ordre total ainsi défini peut alors être utilisé pour résoudre des problèmes de contrôle qui requièrent des prises de décision équitables : la requête la plus ancienne, au sens de l'ordre total défini, étant la requête priviligiée. Il s'agit d'un exemple, parmi baucoup d'autres, où les identités des sites participent au contrôle du système.

L'affectation d'identités distinctes et ordonnées aux sites est généralement réalisée de manière statique, l'identité de chaque site lui étant fournie lors de sa définition. Cette solution peut s'avérer tout à fait satisfaisante lorsque la configuration du système est figée. L'interconnexion de 2 (sous) systèmes pour former un système unique requiert par contre, de s'assurer que les identités de leurs sites respectifs sont différentes, et si tel n'est pas le cas, de faire en sorte qu'elles le soient ; la redéfinition de tout ou partie des sites peut alors être nécessaire.

Nous proposons ici un algorithme réparti, utilisable lors de la génération d'une nouvelle version d'un système réparti, qui affecte des identités distinctes et totalement ordonnées aux sites qui le composent. Initialement les sites n'ont pas d'identité et ne connaissent leur environnement que par les canaux qui les connectent à leurs voisins. Quand l'algorithme termine, chacun des sites est doté d'une identité distincte et est pourvu d'une table de correspondance qui précise pour chacun de ses canaux d'entrée-sortie

l'identité du site voisin auquel ce canal le connecte. l'algorithme transforme ainsi des informations ayant une signification purement locale (les identités de canaux) en informations ayant une signification globale (chaque identité de site a, en effet, une signification unique sur tout le réseau et peut-être exploitée à ce titre).

Le §2 précise les hypothèses ; le §3 présente le principe sur lequel repose l'algorithme et la structure de contrôle sur laquelle il s'appuie ; (un parcours en profondeur du réseau). L'algorithme est présenté au §4. On montre au §5 des situations où la donnée d'identités différentes aux sites permet de résoudre simplement certains problèmes.

2 CONTEXTE ET HYPOTHESES

On considère un système réparti composé de n sites. Il n'existe pas de mémoire commune, partagée par les sites ; ceux-ci sont connectés par un réseau de communication connexe. Les canaux de communication reliant les couples de sites voisins sont bi-directionnels et fiables : ils ne perdent, ni n'altèrent les messages. Les délais de transfert sont arbitraires et finis. Initialement tout site S_k ne connait que ses canaux d'entrée-sortie : $c_{k1}, c_{k2}, ..., c_{kp}$; la désignation d'un canal est purement locale, le type énuméré (local) $canaux_k$ les désigne.

Tout site S_k connait donc le nombre de ses voisins (tout canal le connecte à un voisin) mais ne connait ni son identité, ni la structure du réseau, ni même le nombre total de sites. Nous choisissons, pour des raisons de commodité, d'affecter des identités entières aux sites. Chacun est doté du contexte suivant :

type *canaux* : {*identités (locales) des canaux d'entrée-sortie du site*} ;
var *identité* :**naturel** ;
correspondance :[*canaux*] **de naturel**

Le but de l'algorithme est donc d'affecter, de façon cohérente, des valeurs à l'ensemble des variables locales *identité* et *correspondance* de chacun des sites.

3 PRINCIPE DE L'ALGORITHME

Une façon simple d'assurer l'unicité de chaque identité consiste à doter le système d'une variable globale abstraite :

var *idmax* : **naturel init à** *0*

indiquant la plus grande identité qui, jusqu'à présent a été affectée à un site. Les sites vont alors prélever leurs identités en s'octroyant la valeur suivante de *idmax* et en modifiant *idmax* en conséquence. La variable *idmax* doit être accédée en exclusion mutuelle pour que les identités soient distinctes. Le principe auquel obéit tout site S_k est donc le suivant :

si *le site* S_k *n'a pas d'identité*
alors
    demander la section critique ;
    idmax := *idmax* + *1 ;*
    identité (du site S_k*)* := *idmax ;*
    libérer la section critique
fsi

Le problème se ramène donc :

- à mettre en oeuvre la variable abstraite *idmax* dans le système réparti,
- à assurer qu'elle est accédée en exclusion mutuelle,
- et à assurer que tous les sites prendront une identité.

Il n'est, bien sur, pas possible d'utiliser un algorithme d'exclusion mutuelle pour cela ; en effet d'une part celui-ci nécessite peut-être la connaissance par les sites des identités ou de leur nombre, et d'autre part ne peut pas provoquer la prise d'identité, il pourrait seulement la contrôler.

Provoquer la prise d'identité par chacun des sites peut-être réalisé par un algorithme réparti de parcours de réseau ; un tel algorithme assure en effet que tous les sites seront visités par le parcours [S 83, Che 83, Cha 82, R 87]. Les prises devant être exclusives les uns des autres, un parcours de

réseau séquentiel est tout à fait adéquat : dans un tel parcours, le contrôle matérialisé par un message réalisant la visite, se trouve en effet, à tout instant, localisé sur un seul site ou en transit vers un site [A 85, HR 87, LMT 87, R 87]. La variable globale abstraite *idmax* peut-être implémentée facilement en la faisant transporter par le message qui réalise le parcours : elle sera ainsi, par construction, accédée en exclusion mutuelle.

Le parcours est lancé par un seul site, celui de l'ingénieur-système par exemple. Le lancement effectif sera réalisé, une fois le chargement du système effectué, par la réception par ce site du message *init* émis par l'ingénieur-système.

4 L'ALGORITHME

De nombreux algorithmes de parcours séquentiel (notamment en profondeur) sont possibles [S 83, A 85, HR 88]. Ils se distinguent par des détails algorithmiques et par leurs complexités respectives (en nombre et taille de messages et en temps).

Nous choisissons ici un parcours dans lequel le déplacement du contrôle est matérialisé par la progression des messages *go* et *back* (figure 1) ; ceux-ci sont dotés de 2 champs : *(j, idmax)* ; *j* désigne l'identité de l'émetteur du message et *idmax* a la signification vue précedemment. le message *go(j, idmax)* progresse de site en site en leur donnant leurs identités et en remplissant des entrées des tableaux *correspondance* des sites visités, à l'aide du champ *j*. La fonction *unélementde* permet de choisir un canal vers lequel faire progresser le contrôle. Lorsqu'un site ne peut faire progresser un message *go* vers l'un de ses voisins, il renvoie le contrôle au site duquel il l'a reçu pour la première fois (que nous appelerons son *père*) à l'aide d'un message *back(j, idmax)*. le récepteur d'un tel message fait alors progresser un message *go* vers un de ses voisins auxquels il n'en a pas envoyé et desquels il n'en a pas reçu, s'il en existe. Dans le cas contraire il renvoie *back* à son père ; s'il est lui-même son propre père (on assure par construction que le site ayant reçu le message *init* est son propre père et qu'il est le seul) le parcours est terminé : les sites ont des identités distinctes et les tables de correspondance sont établies. Le site initiateur du parcours connait de plus le nombre *n* de sites du réseau. (Il peut alors le diffuser aux autres sites si cela est nécessaire ; si tel est

le cas il est aisé de construire une arborescence couvrante du réseau, durant le parcours, que l'on exploite ensuite pour réaliser une diffusion en *n-1* messages).

A tout message *go* correspond un message *back* et il y a un message *go* qui transiste sur chaque canal. L'algorithme requiert donc *2e* messages, où *e* est le nombre de canaux. la complexité temporelle est également de *2e*.

5 REMARQUES

Remarque 1. Parcours de réseau = composant logiciel.

D'autres algorithmes de parcours séquentiel peuvent servir de support à l'attribution d'identités. Quel que soit l'algorithme choisi, l'adaptation à faire pour obtenir une affectation d'identités est la même. tout d'abord l'algorithme de parcours doit pouvoir être exprimé en n'utilisant, pour chacun des sites, que la connaissance de ses canaux d'entrée-sortie ; il faut alors greffer les actions de génération de valeurs pour les variables *identité* et *correspondance*.

Comme on le voit, la méthodologie est simple et participe à la définition de composants logiciels (ici le parcours de réseau) qu'il est possible d'enrichir pour réaliser des fonctions particulières [HR 87, HR 88].

Nous présentons maintenant deux situations où la donnée d'identités distinctes est intéressante.

Remarque 2. Réduire le nombre de messages.

Une fois les identités attribuées et les identités des voisins connues, il est possible d'utiliser ces informations de nature globale pour réduire le nombre de messages échangés dans un calcul réparti.

Lorsqu'un site S_i veut diffuser une donnée d, il envoie à ses voisins le message suivant :

$$msg\ (d,\ \{identités\ de\ ses\ voisins\})$$

Lorsque S_j reçoit un tel message : $msg\ (d,\ z)$, il fait progresser vers ceux de ses voisins qui ne sont pas dans z le message :

$$msg\ (d,\ z \cup \{identités\ de\ ses\ voisins\})$$

```
* contexte d'un site
type canaux = {identités des canaux d'entrée-sortie du site};
var identité :naturel ;
    correspondance : [canaux] de naturel ;
    père : naturel ∪ {nil} init à nil ;
    aux := {identités des canaux d'entrée-sortie du site}
    % aux contiendra les canaux sur lesquels la
    correspondance n'a pas été encore établie %
*comportement d'un site
lors de la réception de init % un seul site reçoit un tel message %
début
      identité := 1 ; père := 1 ;
      x := unélémentde (aux) ;
      envoyer go (1, 1) sur canal x
fin
lors de réception de go(j, idmax) sur canal y
début
      correspondance [y] := j ;
      aux := aux - {y} ;
      cas père = nil → % premier message go reçu %
                       père := j ; identité := idmax + 1 ;
                       cas aux ≠ ∅ →x := unélémentde (aux);
                                    envoyer go (identité, idmax + 1)sur canal x
                          aux = ∅ →envoyer back (identité, idmax + 1) sur canal y
                       fcas
         père ≠ nil →% le site a déjà une identité %
                     envoyer back (identité, idmax) sur canal y
      fcas
fin
lors de la réception de back(j, idmax) sur canal y
début
      correspondance [y] := j ;
      aux := aux - y ;
      cas aux ≠ ∅ →x := unélémentde (aux) ;
                   envoyer go (identité, idmax) sur canal x
         aux = ∅ ∧ père ≠ identité →soit x le canal tel que correspondance [x] = père ;
                                    envoyer back (identité, idmax) sur canal x
         aux = ∅ ∧ père = identité →parcours terminé ;
                                    idmax = # sites
      fcas
fin
```

Figure 1 : L'algorithme.

Avec ce mécanisme trés simple, un site ne fait progresser un message vers l'un de ses voisins que si celui-ci ne se trouve pas être sur le chemin, ou n'est pas voisin d'un site sur le chemin, traversé par le message *msg* allant du site diffuseur à lui-même [HPMR 87] ; en effet les sites placés sur un tel chemin ont reçu ou recevront (cela dépend des vitesses des messages sur les canaux) la valeur d .

Remarque 3. Apprendre le réseau.

Les sites une fois dotés d'identités distinctes, il est possible de nommer de manière unique les canaux en identifiant chacun d'eux par le couple (non ordonné) des 2 identités des sites qu'il connecte.

Lorsqu'un site veut apprendre le réseau il diffuse le message composé de 2 champs :

$$msg(\textit{son identité}, \{\textit{identité de ses voisins}\})$$

à ses voisins. A la réception de $msg(j, v)$ un site S_i diffuse vers ses voisins le message :

$$msg(\textit{identité de } S_i, \{\textit{identité de ses voisins}\})$$

s'il ne l'a pas déjà fait. S'il a déjà reçu le message $msg(j, v)$ il ne fait rien. Dans le cas contraire, il le fait progresser vers ses voisins aprés avoir ajouté j à sa variable *sites-connus* (ensemble qui ne contient initialement que son identité) et ajouté $\{(j, x) : \forall\, x \in v\}$ à sa variable *canaux-connus* (ensemble de couples non ordonnés initialisé à $\{(\textit{identité}, x) : \forall\, x = \textit{identité d'un voisin du site}\}$).

Tout site apprendra donc le réseau, c'est-à-dire l'ensemble des sites et celui des canaux. Le problème qui se pose est celui de la terminaison : un site sait qu'il connait tout le réseau, et donc qu'il n'apprendra plus rien, lorsque [B 85] :

$$\forall (x, y) \in \textit{canaux-connus} \Rightarrow x \in \textit{sites-connus et } y \in \textit{sites-connus}$$

Une fois la structure du réseau apprise, les sites peuvent la mettre à profit pour leurs routages et calculs ultérieurs.

6 CONCLUSION

Les parcours parallèles de réseau ont été très étudiés [S 83, Che 83, Cha 82, ZC 87] et utilisés comme élement de base de nombreux algorithmes de contrôle et de calcul réparti [DS 80, CMH 83, BT 87, HR 88].

Nous avons montré ici un intérêt du parcours réparti séquentiel. L'aspect séquentiel assure une visite en exclusion mutuelle de chacun des sites ; si l'on fait transporter les informations critiques par les messages réalisant la visite, la solution de certains problèmes peut s'en trouver grandement simplifiée. Nous avons ici appliqué un tel parcours comme fondement d'un algorithme d'affectation d'identités distinctes et ordonnées aux sites d'un système réparti. Un des intérêts d'une telle affectation dynamique réside dans la facilité offerte lors de la génération de nouvelles versions d'un système ou lors de la reconfiguration aprés pannes ; elle rend en effet la définition des sites indépendantes de valeurs initiales particulières en ce qui concerne leurs identités.

7 REFERENCES

[A 85] AWERBUCH. *A New Distributed Depth-First-Search Algorithm.* Inf. Proc. Letters, Vol. 20, (1985), pp. 147-150.

[B 85] BOUGE L. *Symmetry and Genericity for CSP Distributed Programs.* Rapport de Recherche, LITP, Paris, (Mai 1985) 22 p.

[BT 87] BRACHA G, TOUEG S. *Distributed Deadlock Detection.* Distributed Computing, (1987), pp. 117-130.

[Cha 82] CHANG E. J. H. *Echo Algorithms : Depth Parallel Operations on General Graphs.* IEEE Trans. on Soft. Eng., Vol. SE 8, 4, (1982), pp. 391-401.

[Che 83] CHEUNG T. *Graph Traversal Techniques and the Maximun Flow problem in Distributed Computation.* IEEE Trans. on Soft. Eng., Vol. SE 9, 4, (1983), pp. 504-512.

[CHM 83] CHANDY K., MISRA J., HAAS L., *Distributed Deadlock Detection.* ACM TOCS, Vol. 1, 2, (May 1983), pp. 144-156.

[DS 80] DIJKSTRA E. W. D., SCHOLTEN C. S. *Termination Detection For Diffusing Computations.* Inf. Proc. Letters, Vol. 11, 1, (August 1980), pp. 217-219.

[HPMR 87] HELARY J. M , PLOUZEAU N., MADDI A., RAYNAL M. *Parcours et Apprentissage dans un Réseau de Processus communicants.* TSI, Vol. 6, 2, (1987), pp. 127-139.

[HR 87] HELARY J. M, RAYNAL M. *Depth-First Traversal and Virtual Ring Construction in Distributed System.* Rapport de Recherche INRIA, n° 704, (juillet 1987), 15 p.

[HR 88] HELARY J. M, RAYNAL M. *Synchronisation et Contrôle des Systèmes et des Programmes Répartis.* Eyrolles Ed., (1988), 210 p.

[L 78] LAMPORT L. *Time, Clocks and the Odering of Events in a Distributed System.* Comm. ACM, Vol. 21, 7, (1978), pp. 558-565.

[LMT 87] LAKSMANAN K.B., MEENAKSHI N., THULISARAMAN K. *A Time-Optimal Message Efficient Distributed Algorithm for Depth First Search.* Inf. proc. Letters, Vol. 25, (1987), pp. 103-109.

[NT 87] NEIGER G., TOUEG S. *Substituting for Real Time and Common Knowledge in Asynchronous Distributed Systems.* Proc. 6th ACM Symposuim on PODC, Vancouver, (1987) pp. 281-293.

[R 87] RAYNAL M. *Systèmes Répartis et Réseaux : Concepts, Outils et Algorithmes.* Eyrolles Ed., (1987), 200 p.

[RA 81] RICART G., AGRAWALA A. K. *An Optimal Algorithm for Mutual Exclusion in Computer Networks.* Comm. ACM, Vol. 24, 1, (Jan. 1981), pp. 9-17.

[S 83] SEGALL A. *Distributed Network Protocols.* IEEE Trans. on Inf. Theory, Vol. IT 29, 1, (Jan. 1983), pp. 23-35.

[ZC 87] ZHU Y., CHEUNG T. Y. *A New Distributed Breadth-First-Search Algorithm.* Inf. Proc. Letters, Vol. 25, (1987), pp. 329-333.

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