Parcours et apprentissage dans un réseau de processus communicants

Jean-Michel Hélary, Aomar Maddi, Noël Plouzeau, Michel Raynal

To cite this version:

Jean-Michel Hélary, Aomar Maddi, Noël Plouzeau, Michel Raynal. Parcours et apprentissage dans un réseau de processus communicants. [Rapport de recherche] RR-0543, INRIA. 1986. inria-00076011

HAL Id: inria-00076011
https://inria.hal.science/inria-00076011

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

CENTRE DE RENNES
IRISA

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° 543

PARCOURS ET APPRENTISSAGE DANS UN RÉSEAU DE PROCESSUS COMMUNICANTS

Jean-Michel HELARY
Aomar MADDI
Noël PLOUZEAU
Michel RAYNAL

Juillet 1986

Jean-Michel HELARY

Aomar MADDI

Noël PLOUZEAU

Michel RAYNAL

PARCOURS ET APPRENTISSAGE DANS UN RESEAU DE PROCESSUS COMMUNICANTS

PAPIER RECUPERÉ ET RECYCLE

PARCOURS ET APPRENTISSAGE DANS UN RESEAU DE PROCESSUS COMMUNICANTS

J.M. Hélary, A. Maddi, N. Plouzeau, M. Raynal
I.R.I.S.A - Av. du Général LECLERC - 35042 RENNES

Résumé

Les algorithmes distribués, qui constituent les composants de base des applications et des systèmes répartis, sont formés d'un réseau de processus communicant par messages. Un grand nombre d'entre eux, dont le rôle est de réaliser une fonction de contrôle, nécessitent la mise en œuvre de parcours du réseau par des messages ou l'apprentissage par un processus donné d'une information de nature globale.

Cet article examine ces deux problèmes. A partir de l'analyse d'un problème simple (la diffusion d'une information) une technique générale visant à réduire le nombre de messages échangés est introduite. Le contrôle des transferts introduit est appliqué aux techniques de parcours canoniques de réseau qui sont ensuite étudiées et à un paradigme relatif à l'apprentissage d'une connaissance globale par un processus donné. La généralité du contrôle des transferts, des modes d'exploration et de la technique d'apprentissage proposés est illustrée par des exemples. Outre leur caractère général, les modes de parcours et le paradigme d'apprentissage sont intéressants par leur originalité et le nombre de messages qu'ils nécessitent (et qu'une analyse de complexité étudie).

Abstract

Distributed algorithms (which are the fundamental parts of distributed systems) are made of processes who interact by the only mean of messages carried by a communication network. Most of these algorithms have to perform some control function and thus need to traverse the network with messages or to learn some global information.

This paper focus on these two problems. A general method is presented from the study of a simple problem (i.e. information diffusion). The aim of this method is to reduce the message counts of the processing. A transfer control strategy is added to the canonical network traversal methods (which are also studied) and to the paradigm of global knowledge learning performed by some process. The general presentation of transfer control, exploration methods and learning technics is then enlighted through examples. Apart from their generality, exploration methods and learning paradigms are worth noting because of their originality and the low message count that they permit (these counts are evaluated in a complexity study).

1. Introduction

Un algorithme distribué est constitué d'un ensemble fini de processus qui communiquent entre eux par échange de messages uniquement. Le support physique peut être un réseau constitué de sites interconnectés par des lignes de communication. Sans perte de généralité nous confondrons par la suite les termes *processus* et *site*. Un tel réseau de processus peut être modélisé par un graphe $G=(X,U)$, non orienté si les lignes sont bidirectionnelles, et connexe (sinon il y aurait plusieurs sous-réseaux ne communiquant pas entre eux). Pour participer à un algorithme distribué, chaque processus exécute un algorithme séquentiel qui lui est propre (données et code locaux), et qui peut s'exprimer sous forme « événementielle » (on trouvera dans [RAY 85] de nombreux algorithmes distribués présentés sous cette forme) :

```
lors de  événement
faire
   action
fait
```

Nous pouvons distinguer deux sortes d'événements :

- ceux qui sont produits par le processus lui-même : nous les appellerons « *événements spontanés* » ou « *décisions* »,
- ceux qui sont causés par d'autres processus (par exemple : la réception de messages).

Les premiers interviennent généralement lors du lancement de l'exécution d'un algorithme distribué par un ou plusieurs processus (ils peuvent souvent être « simulés » par une réception fictive de messages que le processus s'enverrait à lui-même). Une telle structuration des événements existe dans certains langages notamment dans le langage de description d'applications réparties *ESTELLE* (on peut en trouver une présentation dans [LIN 85]).

L'action peut être généralement découpée selon deux composantes :

- un traitement local,
- l'envoi de messages vers les processus voisins.

Tout message est porteur d'une information (éventuellement réduite à un pur signal). A la réception d'un message, l'information apportée par celui-ci, conjointement à la connaissance locale du processus récepteur, constituent les données de l'action. Celles-ci sont utilisées aussi bien pour le traitement (qui agit sur le contexte local) que pour décider de l'émission de messages, vers quels

processus, et porteurs de quelle information (établie à partir de la connaissance locale au moment de l'émission). Le rôle de l'information transportée par un message est double :

1) rôle de données,
2) rôle de contrôle.

Par la première fonction, l'information permet au processus d'enrichir sa connaissance locale afin d'effectuer le traitement local tandis que le second rôle vise à permettre aux processus de décider vers quels voisins il y lieu d'émettre tel message (implémentation locale du contrôle distribué du déroulement de l'algorithme). L'élaboration de l'information à émettre peut relever des deux fonctions de l'information reçue (il va de soi qu'une partie de l'information permettant au processus d'exercer un tel contrôle peut aussi résider dans le contexte local, comme par exemple dans le langage *CSP* [HOA 78]). Nous appellerons *connaissance de contrôle* les informations permettant à un processus d'interagir avec les autres processus de l'algorithme, et *information de contrôle* tout champ de l'information transportée par un message, grâce auquel le récepteur du message peut mettre à jour sa connaissance de contrôle. L'exploitation d'une connaissance de contrôle – qui peut, entre autres choses, avoir pour but d'éviter au maximum la transmission de messages inutiles – nécessite en général des hypothèses sur la connaissance de l'environnement que possède chaque processus plus fortes que la simple connaissance des lignes (nécessaire pour tout échange) qui le connectent à d'autres processus.

En effet lorsque la connaissance de son environnement par un processus se limite à des noms locaux (par exemple des ports ou des canaux auxquels il est connecté) cette connaissance de contrôle ne peut rester que locale vue la portée de ces noms. Si par contre la connaissance de son environnement par un processus est donnée par des noms globaux (faisant intervenir par exemple l'identité des processus), elle est exploitable de manière beaucoup plus fine pour le contrôle global de l'algorithme distribué (contrairement à un nom local les identités de processus, si elles sont toutes distinctes, constituent des informations de nature globale exploitable par tout processus [SIL 81, WEG 83, WAT 83]). La connaissance de l'environnement possédée par un processus peut alors s'enrichir au cours du déroulement de l'algorithme et le contrôle exercé localement par ce processus sur ce déroulement peut devenir important.

Dans le paragraphe suivant, nous introduisons – à l'aide d'un exemple très simple de protocole (diffusion d'une information) – un principe de contrôle du transfert de messages dans un réseau de processus communicants, où l'environnement de chacun des processus est défini par les identités de ses voisins. Dans les troisième et quatrième paragraphes, nous appliquons ce principe à deux problèmes de base (qui recouvrent de nombreuses applications) :

- la construction d'une arborescence recouvrante (parcours de réseaux)
- l'interrogation du réseau par un processus (apprentissage d'une connaissance globale).

Chacun de ces deux problèmes est étudié avec deux schémas de contrôle différents.
L'approche de l'étude se place du point de vue « algorithmique et système » (définition de schémas de contrôle et d'exécution) et non du point de vue « langage » au sens de [LIS 83] (définition de constructions linguistiques).

2. Contrôle de la diffusion d'une information

2.1 Hypothèses

Dans ce paragraphe, nous nous plaçons dans le cadre des hypothèses suivantes :

- sur le réseau :
  - le graphe des communications est connexe et sa topologie est quelconque,
  - les messages ne sont pas perdus (tout message émis sera délivré au bout d'un temps arbitraire, mais fini) et ne sont pas altérés .
- sur la connaissance de l'environnement :
  - ou bien chaque processus ne connaît que les noms des canaux auxquels il est connecté (CL1),
  - ou bien chaque processus connaît les identités de ses voisins (CL2) (les identités des processus étant toutes différentes).

    Un processus ne connaît donc ni le nombre total de processus ni la structure globale du réseau.

2.2 Problème de la diffusion

Soit P_{init} un processus, possédant une information m. Celle-ci doit être communiquée à tous les autres processus du réseau. L'idée de base, très simple, est fondée sur la technique de la vague [SCH 85] : P_{init} envoie à chacun de ses voisins un message transportant m. A la réception d'un tel message, un processus le réémet à son tour à chacun de ses voisins, et ainsi de suite.

Chaque processus P_i exécute donc l'algorithme suivant :

```
lors de réception de m sur canal c_in
faire
      mémoriser m dans le contexte local;
      ∀ c ∈ canaux_i-c_in : envoyer m sur c
fait
```

P_{init} lance l'algorithme en simulant la réception de m. Le contexte local de P_i, dans cet algorithme, est le suivant :

- une structure de données permettant la mémorisation de m
- *canaux*$_i$: `ensemble de lignes` (* reliant P_i à ses voisins *).

Comme on le voit, cet algorithme est totalement exempt de contrôle, puisque la seule information transportée – à savoir m – est une information « de données ». En particulier, cet algorithme ne se termine pas.

Nous allons examiner deux contrôles : le premier effectué à la réception va assurer la terminaison, le second effectué à l'émission va diminuer le nombre de messages échangés.

2.3 Contrôle assurant la terminaison

Un premier contrôle, n'utilisant que le contexte local des processus – et donc applicable dans le cadre de l'hypothèse **(CL1)** – peut être introduit afin d'assurer la terminaison. Chaque processus P_i est alors doté d'une variable booléenne *reçu*$_i$, initialisée à **faux**. Le contrôle consiste à assurer que P_i ne réémet m que lorsqu'il reçoit cette information pour la première fois **[SEG 83]**.

Le texte de P_i devient :

```
lors de réception de m sur c_in
faire
      si ¬reçu_i alors
          reçu_i:=vrai;
          mémoriser m dans le contexte local;
          ∀ c ∈ canaux_i-c_in : envoyer m sur c
      fsi
fait
```

Le contrôle est donc exercé ici *a posteriori* ; autrement dit, sachant qu'il est inutile qu'un processus reçoive plus d'une fois la même information, on ne fait qu'examiner, au niveau de la réception, si ce fait s'est déjà produit ; le récepteur ignore alors le message. Par ailleurs, l'information de contrôle reste purement locale et n'est jamais transmise.

2.4 Contrôle à l'émission des messages

Il serait donc plus intéressant d'introduire un contrôle plus fin, qui s'exercerait *a priori* ;

autrement dit, un processus P_i s'abstiendrait d'envoyer m à P_j dès lors qu'il saurait que P_j a reçu (ou va recevoir) cette information – de lui-même ou d'un autre processus. La mise en œuvre d'un tel contrôle implique qu'un processus, quel qu'il soit, sache discriminer ses voisins par leur identité, et donc que l'on se place dans le cadre de l'hypothèse **(CL2)**. D'autre part, pour qu'un processus apprenne une connaissance de contrôle qu'il ne possède pas initialement, il faudra véhiculer, dans les messages échangés, une information de contrôle. Ceci étant, le contrôle suivant peut être proposé **[HMR 86a]** ; on adjoint au contexte local d'un processus P_i la constante :

$voisins_i$: ensemble de noms = ensemble des identités des voisins de P_i (connaissance locale initiale).

On ajoute au message m un champ z qui désigne un ensemble de noms (information de contrôle). P_{init} transmet à chacun de ses voisins :

- l'information m (donnée),
- son identité et celle de ses voisins (contrôle).

A la réception d'un message, P_i enrichit l'information de contrôle reçue (champ z) – qui contient un sous-ensemble des processus ayant reçu ou qui vont recevoir la donnée m – avec sa propre connaissance locale, qui est celle de ses interlocuteurs. Il ne réémet alors le message qu'à ceux de ses voisins n'appartenant pas à z, et ce message transporte :

- l'information m (donnée),
- l'information de contrôle enrichie par P_i.

On obtient donc le texte suivant pour P_i :

```
lors de réception de (m,z)
faire
      si ¬reçu_i alors
          reçu_i := vrai;
          mémoriser m dans le contexte local;
          soit y = voisins_i - z;
          (* y contient les noms des voisins non encore informés, *)
          (* tel que le perçoit P_i.                               *)
          si y ≠ Ø alors
              ∀ k ∈ y : envoyer (m,z∪y) à P_k
          fsi
      fsi
fait
```

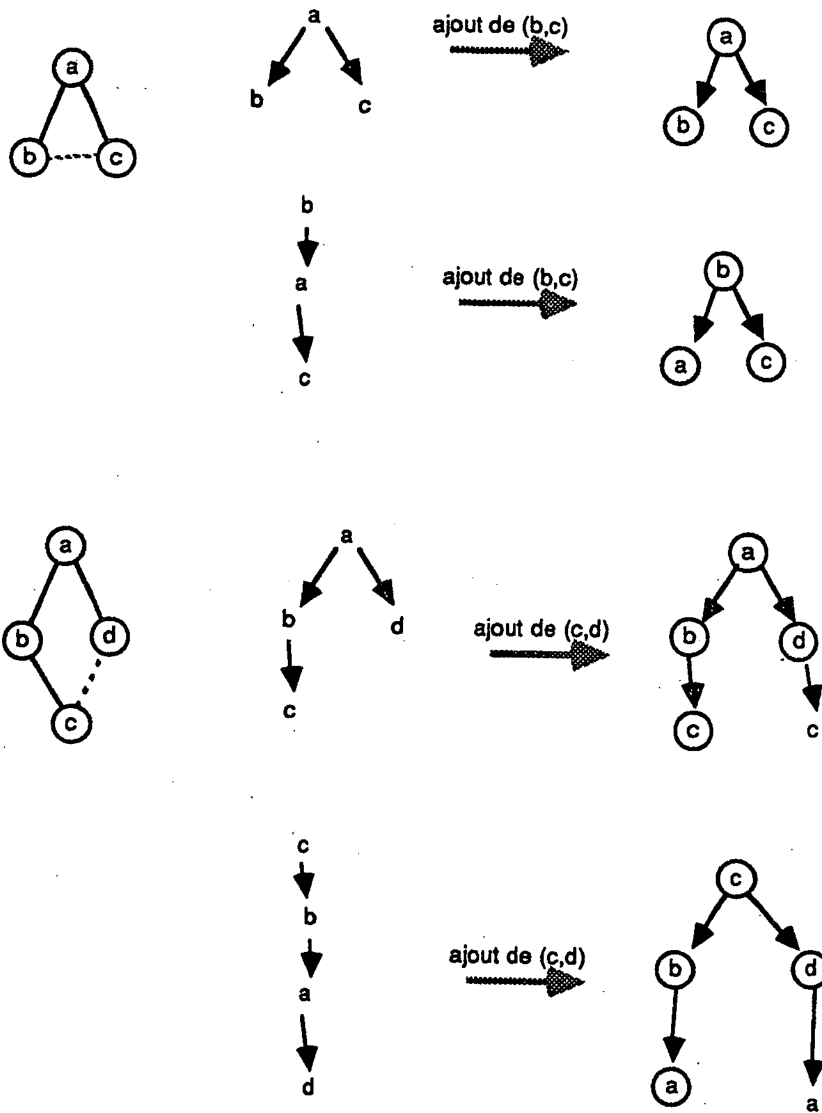
P_{init} initialise l'algorithme en simulant la réception de $(m,\{init\})$. L'algorithme se termine mais aucun processus ne peut savoir s'il a lui-même terminé ni, *a fortiori*, si l'algorithme est terminé.

2.5 Analyse

Le prix à payer pour la mise en œuvre de ce contrôle est dû à l'hypothèse **(CL2)**. Or, celle-ci peut être assurée à partir de **(CL1)**, grâce à un protocole simple : deux messages par canal (un dans chaque sens) permettent l'échange des identités entre tout couple de processus voisins **[SEG 83]**. De plus, l'hypothèse **(CL2)** peut être maintenue en cas de changement de configuration du réseau (panne ou reconnexion). Le protocole d'apprentissage des identités des voisins peut donc être exécuté une fois pour toutes, par exemple lors de l'initialisation du réseau. Insistons sur le fait que la connaissance de l'environnement par chacun des processus reste locale : ni la topologie du réseau, ni l'identité des processus non voisins, ni même le nombre total de processus n'est connu ; ceci permet de résister plus facilement aux changements de configuration : seuls les processus dont le voisinage a été modifié devront être recompilés. Ayant ainsi analysé le prix à payer pour la mise en œuvre de ce contrôle, quel en est le gain ? La progression des messages, dans le réseau, définit une arborescence de contrôle dont chaque nœud est étiqueté par l'identité du processus recevant le message :

- la racine en est P_{init},
- lors de la réception d'un message par P_i, un nœud d'étiquette i est créé dans l'arborescence ; il a pour père le nœud d'étiquette j, correspondant à l'émetteur P_j de ce message ;
  - si c'est la première réception, P_i réémet le message vers un sous-ensemble $succ_i$ de ses voisins, et les nœuds correspondants deviennent ses fils ,
  - sinon, le nœud est une feuille.

Quelque soit P_i, il existe au moins un nœud d'étiquette i, et parmi tous les nœuds d'étiquette i, au plus un n'est pas une feuille. Le contrôle proposé s'exprime donc de la manière suivante : un processus ne transmet l'information qu'à ceux de ses voisins qui ne sont pas ses antécédents, ou les voisins de ses antécédents, dans l'arbre de contrôle.



Les nœuds entourés symbolisent la première occurence d'un message sur le processus

Figure 1

Pour un réseau constitué de n processus, désignons par C_n (resp. K_n) la complexité, en nombre de messages, de l'algorithme **2.4** (resp. **2.3**) ; le gain est alors K_n-C_n. Le nombre de lignes e varie entre $n-1$ (arbre) et $n(n-1)/2$ (réseau complet). Si $e=n-1$, on a évidemment $C_n=K_n=n-1$, et dans ce cas le gain est nul. Lorsque e est incrémenté d'une unité, l'ajout de cette nouvelle ligne – qui crée un nouveau cycle dans le réseau – augmente la valeur de K_n de 1 ou 2

unités (selon la configuration et la vitesse des messages). Par contre, la variation de C_n e[...]
maximum, de :

0 si le cycle créé est de longueur *3*,

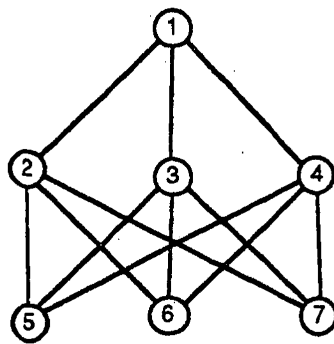
1 si le cycle créé est de longueur *4*,

2 si le cycle créé est de longueur supérieure ou égale à *5*.

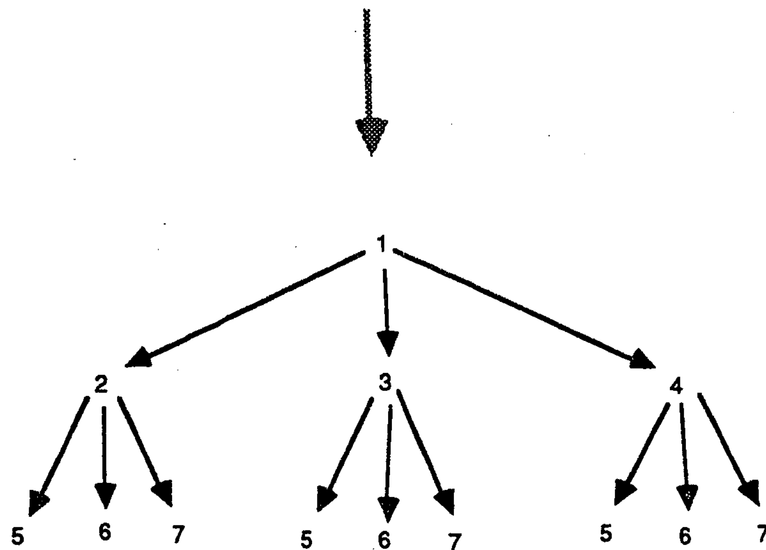
Si on note $\gamma_3(n)$ le nombre maximum d'arêtes d'un graphe non orienté à n sommet[...]
possédant pas de cycles de longueur *3*, on a la propriété suivante : dès que e est supérieur ou [...]
$\gamma_3(n)$, l'ajout de toute nouvelle ligne n'engendre aucune augmentation de C_n tandis [...]
$K_n$augmente de une à deux unités (figure 1). Sachant que $\gamma_3(n)$ vaut $n^2/4$ si n est pair et $(n^2$[...]
si n est impair, on en déduit que le gain K_n–C_n est quadratique en n pour les réseaux denses. [...]
un réseau complet e vaut exactement $n(n-1)/2$ et on a $C_n=n-1$, $K_n=(n-1)^2$.

Une étude plus détaillée de la complexité C_n en fonction de la configuration du réseau re[...]
faire ; remarquons toutefois que la complexité est linéaire aux deux extrémités (pour e valant [...]
près $n-1$ et pour e valant à peu près $n(n-1)/2$), tandis qu'elle est quadratique, de l'ordre de [...]
dans la configuration « défavorable » suivante (figure 2) :

V1 : voisins de P1



Graphe des communications



Arbre des messages transmis lors d'une diffusion depuis P1

Figure 2

On notera V_x l'ensemble des voisins du processus x, le processus x n'appartenant pas à V_x.

$U=\{P_{init}\}\times V_{init} \cup V_{init}\times W_{init}$
où $W_{init}=X-V_{init}-\{P_{init}\}$
et $card(V_{init})=n \; div \; 2$

En tout état de cause, la complexité C_n est au pire en $O(e)$.

3. Contruction d'une arborescence recouvrante

3.1 Le problème

Etant donné un réseau modélisé par un graphe connexe $G=(X,U)$, un des sites du réseau désire construire une arborescence recouvrante de G, dont il sera la racine ; à l'issue de cette construction, tout site connaîtra son père et ses fils dans l'arborescence. Cette information définit un routage ascendant de tout site vers le site racine, ce qui permet à ce dernier d'être interrogé ou informé par tout site du réseau.

Remarque : Les algorithmes que nous allons développer pourraient facilement être adaptés de telle sorte que le routage descendant soit connu, c'est à dire que chaque site P_i connaisse son successeur sur le chemin unique allant de P_i à P_j (ou l'absence d'un tel successeur si P_j n'est pas descendant de P_i dans l'arborescence). Nous ne le ferons pas ici, notre but étant essentiellement de présenter une mise en œuvre du principe de contrôle des transferts de connaissance. Un tel routage une fois établi permet d'assurer des diffusions depuis la racine en $n-1$ messages.

Nous examinons successivement deux types d'explorations permettant de résoudre le problème : l'exploration *parallèle* (qui est une exploration en largeur d'abord si tous les messages transitent sur les canaux à la même vitesse), application directe de la technique de diffusion contrôlée du paragraphe précédent, et l'exploration *séquentielle* (ou en profondeur), offrant un contrôle plus rigoureux. Nous supposons toujours que **(CL2)** est vérifiée.

3.2 Exploration parallèle

3.2.1 Le principe

Le processus P_{init} qui souhaite lancer la construction d'une arborescence diffuse dans le réseau un message d'exploration, selon la technique de diffusion contrôlée exprimée au §2 (schéma de l'algorithme **2.4**). A cette diffusion va s'ajouter une technique d'acquittement, permettant d'une part à P_{init} d'apprendre que l'algorithme est terminé, d'autre part d'assurer une

construction correcte de l'arborescence ; un message d'acquittement, transportant une information de contrôle appropriée, sera renvoyé par un processus P_i en réponse à toute réception de message d'exploration, selon les modalités qui suivent.

Pour obtenir une arborescence recouvrante, il est nécessaire :

i) que tout processus autre que P_{init} ait un prédécesseur unique,

ii) que deux processus distincts aient des ensembles de successeurs disjoints.

La condition *i)* est facilement réalisée : le prédécesseur de P_i est défini comme étant l'émetteur du premier message d'exploration atteignant P_i.

Le premier message reçu est, au sens de la diffusion contrôlée, le seul pris en compte par P_i ; c'est lors de la réception de celui-ci que P_i propage l'exploration vers un sous-ensemble de l'ensemble de ses voisins, établi à partir de l'information de contrôle. Ce sous-ensemble est alors interprété par P_i comme étant l'ensemble de ses successeurs. Or, malgré le contrôle *a priori* de la diffusion, un même processus peut être atteint par des messages d'exploration provenant d'émetteurs distincts, et par conséquent apparaître dans plusieurs ensembles $succ_i$.

Pour satisfaire la condition *ii)*, on adopte le protocole suivant [SEG 83]. Tout processus P_i, lorsqu'il reçoit pour la première fois un message d'exploration venant de P_j, établit son prédécesseur (P_j), et l'ensemble de ses successeurs (les voisins vers lesquels il propage l'exploration) ; il envoie à P_j un message d'appartenance soit immédiatement si l'ensemble de ses successeurs est vide, soit dès qu'il a reçu les réponses de tous ses successeurs. La réception de ce message par P_j est donc une confirmation de l'appartenance de P_i à l'ensemble de ses successeurs. Lors de réceptions ultérieures de messages d'exploration, P_i renverra immédiatement à leur émetteur une information lui indiquant de ne pas l'inclure dans ses successeurs. Cette information sera véhiculée par un message de réponse.

La réception par P_{init} d'une réponse de chacun de ses voisins l'informe de la terminaison de l'algorithme (plus aucun message relatif à cet algorithme ne circule). Nous retrouvons là une technique générale d'exploration d'un arbre de contrôle utilisée dans des algorithmes de terminaison distribuée [DIS 80,FRR 82,MIS 82], dans des algorithmes de détection d'interblocage [CMH 83], dans des algorithmes de calcul distribué sur des graphes [CHA 82, CHE 83]; elle est ici couplée avec le principe de diffusion contrôlée, introduit au §2.4.

3.2.2 Les messages

Deux types de message sont donc nécessaires. Ils se présentent sous la forme suivante :

• *(explorer,z)* message d'exploration avec l'information de contrôle z ayant la même signification qu'au §2,

• *(acquitter,x)* où x permet au récepteur de savoir s'il doit considérer l'émetteur comme un de ses successeurs dans l'arborescence (*x=terminé*) ou non (*x=déjàvu*).

3.2.3 Contexte local d'un processus

Chaque processus P_i est doté du contexte local suivant :

```
const
   voisinsi  :   ensemble de noms
```

voisins$_i$: **ensemble de** noms (c'est l'ensemble des identités des voisins de P_i). Cet ensemble définit le voisinage de P_i, qui est sa seule connaissance globale sur le réseau.

var

pred$_i$: nom **init nil**

succ$_i$: **ensemble de** noms **init** ∅
Ces deux variables servent à mémoriser les liens de P_i dans l'arborescence construite.

atteint$_i$: **booléen init faux**, devenant vrai dès que P_i reçoit un message *explorer*.

nbacq$_i$: **entier init** 0, qui est une variable de contrôle permettant à P_i de mémoriser le nombre d'acquittements attendus.

3.2.4 Texte de P_i

```
lors de réception de (explorer,z)  depuis  Pj
faire
     si atteinti alors envoyer (acquitter,déjàvu) à  Pj
     sinon
         atteinti := vrai;
         predi := j;
         soit y = voisinsi - z;
         cas
             y =∅ → envoyer (acquitter,terminé) à  Pj
             y≠∅ →
                 ∀k ∈ y : envoyer (explorer,z∪y) à Pk
                 nbacqi := card(y);
                 succi  := y
         fcas
     fsi
fait
```

```
lors de réception de (acquitter,x) depuis Pj
faire
      si x = déjàvu alors succi := succi - {j} fsi;
      nbacqi := nbacqi - 1;
      si nbacqi = 0 alors
          cas
              i ≠ init→ envoyer (acquitter,terminé) à Ppredi
              i = init→ (* l'algorithme est terminé *)
          fcas
      fsi
fait
```

P_{init} lance l'algorithme en simulant la réception de (*explorer*,{*init*}).

3.2.5 Complexité

En ce qui concerne la complexité de la taille du champ *z*, le nombre maximal d'identités dans *z* est *n* ; cette complexité est donc linéaire.

En ce qui concerne la complexité en nombre de messages échangés nous avons ici C_n messages *explorer* (l'algorithme étant fondé sur la technique de diffusion contrôlée du §2.4 avec *m = explorer* ; ce nombre C_n est inférieur ou égal à $2e$ car il y au plus un message *explorer* émis dans chaque sens sur chaque canal). De plus tout message *explorer* engendre exactement un message *acquitter*. La complexité est donc $2C_n$ messages. Rappelons que C_n est compris, selon la densité du réseau, entre $n-1$ et $O(n^2)$; la valeur $n-1$ étant atteinte pour les deux densités extrêmes : l'arbre et le réseau complet.

3.3 Exploration en profondeur

Nous présentons maintenant une technique d'exploration qui améliore la précédente, en ce sens que tout processus du réseau sera atteint une et une seule fois par l'exploration, ce qui permet de construire l'arborescence sans remise en cause des ensembles de successeurs, et ceci avec une complexité linéaire en nombre de messages (c'est donc, dans le contexte distribué, l'équivalent d'un algorithme « glouton »). Ce résultat est obtenu sans augmenter la connaissance du réseau possédée par chaque site, mais grâce à un renforcement de l'information de contrôle transportée par les messages. L'algorithme obtenu améliore les algorithmes d'exploration en profondeur déjà proposés **[CHE 83, AWE 85]** dont la complexité en nombre de messages est $O(e)$ et la complexité en temps respectivement en $O(e)$ pour le premier et $O(n)$ pour le second.

3.3.1 Principe

Le processus P_{init} qui souhaite lancer la construction de l'arborescence en informe un de ses voisins et le mémorise comme successeur. Lorsqu'un processus P_i reçoit un message d'exploration (au premier message reçu il mémorise l'émetteur P_j comme étant son père), trois cas peuvent se présenter :

cas *i)* P_i sait que tous les autres processus sont déjà dans l'arborescence. Il va alors informer tous les autres sites que l'algorithme est terminé, en initialisant une diffusion le long de l'arborescence construite (par l'envoi d'un message à son père).

cas *ii)* P_i sait que des processus ne sont pas encore dans l'arborescence, mais que tous ses voisins y sont. Il va alors renvoyer un message d'acquittement à son père.

cas *iii)* P_i sait qu'il a des voisins non encore atteints. Il sélectionne alors l'un d'entre eux, le mémorise comme successeur et l'informe par l'envoi d'un message d'exploration.

Lorsqu'un processus reçoit un message d'acquittement (il est alors nécessairement atteint), il agit comme en *ii)* ou *iii)* ci-dessus (le cas *i)* ne pouvant se produire dans cette situation). Enfin, lorsqu'un processus apprend que l'algorithme est terminé, il répercute cette information vers tous ses autres voisins de l'arborescence.

3.3.2 Les messages.

La mise en œuvre de cet algorithme nécessite donc trois types de messages, lesquels sont porteurs d'informations de contrôle permettant au processus récepteur d'acquérir une connaissance suffisante pour exercer le contrôle décrit dans les cas *i), ii), iii)*. Les messages sont donc :

• *(explorer,z,s)* où z et s sont des ensembles d'identités de processus, ayant la signification suivante :

- z désigne l'ensemble des identités des processus déjà atteints,
- s désigne l'ensemble des identités des voisins non atteints des processus déjà atteints.

Ainsi, $atteint(P_i)$ étant un prédicat valant *vrai* si et seulement si P_i est dans l'arborescence (autrement dit si P_i a reçu un message *explorer*), les relations suivantes seront toujours vérifiées (invariants) :

(1) $k \in z \Leftrightarrow atteint(P_k)$

(2) $k \in s \Leftrightarrow \neg\, atteint(P_l) \wedge (\exists k : atteint(P_k) \wedge (P_l, P_k) \in U)$

Un message *explorer* est émis à l'initialisation par P_{init}, puis par tout processus dans le cas *iii)*.

- *(rebrousser,z,s)* où *z* et *s* ont la même signification que pour *explorer*. Un tel message est émis dans le cas *ii)*.
- *(terminé)* est un message qui porte l'information de terminaison (émis dans le cas *i)*).

3.3.3 Exploitation de l'information de contrôle

Les expressions (1) et (2) étant des invariants – nous verrons que c'est le cas plus loin – l'exploitation que peut faire un processus P_i de l'information de contrôle *(z,s)* est la suivante :

cas

$voisins_i \subseteq z \rightarrow$

Tous les voisins de P_i sont atteints ;

si $s=\varnothing$ **alors**

P_i peut conclure qu'il est le dernier à être atteint (en effet, tous ses voisins sont atteints et tous les voisins de tout processus atteint sont atteints ; donc, de proche en proche, tous les autres processus sont atteints).

sinon

il existe des processus non atteints, mais pas parmi les voisins de P_i : c'est le cas *ii)*.

fsi

$voisins_i \cap z \neq \varnothing \rightarrow$ il existe des voisins de P_i non atteints : c'est le cas *iii)*.

fcas

3.3.4 Mise à jour de l'information de contrôle

La mise à jour de l'information de contrôle doit donc être guidée par le maintien des invariants (1) et (2). Par conséquent :

- Initialement, P_{init} envoie à l'un de ses voisins P_j : $(explorer,\{init\},voisins_{init}-\{j\})$
- Lorsque P_i reçoit *(explorer,z,s)* depuis P_j alors :

  cas *i)* $(voisins_i \subseteq z \wedge s=\varnothing)$: envoi à P_j du message *terminé* (pas de mise à jour à faire)

cas *ii)* ($voisins_i \subseteq z \wedge s \neq \emptyset$) : P_i est atteint, donc : $z \leftarrow z \cup \{i\}$; s ne change pas puisque $voisins_i \subseteq z$. L'information de contrôle *(z,s)* envoyée à P_j (par un message *rebrousser*) vérifie donc bien (1) et (2).

cas *iii)* ($\neg(voisins_i \subseteq z)$). Soit $y = voisins_i - z$ et $k \in y$. P_i est atteint, donc : $z \leftarrow z \cup \{i\}$; P_i envoie à P_k un message *explorer*, donc les voisins non atteints de P_i doivent entrer dans s : $s \leftarrow s \cup y - \{k\}$.

- Lorsque P_i reçoit *(rebrousser,z,s)* depuis P_j alors :

cas *i)* impossible (pour que ce message soit émis il faut que $s \neq \emptyset$).

cas *ii)* z et s ne changent pas et l'information de contrôle est renvoyée au père de P_i par un message *(rebrousser,z,s)*.

cas *iii)* P_i a été précédemment atteint et, à ce moment là, tous les voisins non atteints de P_i ont été mémorisés dans s. Depuis, et jusqu'à l'étape examinée ici, chaque fois qu'un voisin de P_i a été atteint (depuis P_i, ou depuis un autre processus), il a été ôté de s. On a donc, à l'étape courante : $y \subset s$. Par conséquent, la mise à jour est la suivante : z ne change pas, $s \leftarrow s - \{k\}$ où P_k est un voisin non atteint de P_i ($k \in y$) selectionné par P_i.

3.3.5 Contexte local et texte d'un processus P_i

Pour un processus P_i donné, par rapport à l'algorithme **3.2**, les variables $atteint_i$ et $nbacq_i$ ne sont plus nécessaires. Le reste est sans changement. Le texte de P_i est le suivant :

```
lors de réception de (explorer,z,s) depuis Pj
faire
     predi:=j;
     soit y=voisinsi-z;
     cas
         y=∅ et s=∅ → envoyer terminé à Pj
         y=∅ et s≠∅ →
              envoyer (rebrousser,z∪{i},s) à Pj
         y≠∅ →
             soit k=élément(y);
             succi:=succi∪{k};
             envoyer (explorer,z∪{i},s∪y-{k}) à Pk
     fcas
fait
```

```
lors de réception de (rebrousser,z,s) depuis Pj
faire
      soit y=voisinsi-z;
      cas      y=∅ → envoyer (rebrousser,z,s) à Ppredi
               y≠∅ →
                    soit k=élément(y);
                    succi:=succi∪{k};
                    envoyer (explorer,z,s-{k}) à Pk
      fcas
fait

lors de réception de (terminé) depuis Pj
faire
      ∀k∈succi∪{predi}-{j} envoyer terminé à Pk;
      (* L'algorithme est terminé pour le processus Pi *)
fait
```

P_{init} lance l'algorithme en simulant la réception d'un message *(explorer,∅,∅)*. L'algorithme est terminé dès que tous les processus ont reçu un message *terminé* (les processus savent que cet événement se produira, mais ils ne savent pas quand).

3.3.6 Heuristiques de parcours

Dans le modèle précédent le choix du voisin vers lequel P_i fait progresser le message *explorer* est arbitraire : il s'agit d'un élément quelconque de l'ensemble y. Le choix peut être précisé en fonction du problème résolu à l'aide du modèle. A la place de la fonction *élément* on peut, par exemple, utiliser la fonction *maximum* si on désire faire un parcours en visitant les processus de plus grande identité d'abord [HMR 86b]. Des heuristiques de parcours peuvent être ainsi formulées en fonction des problèmes particuliers à résoudre.

3.3.7 Complexité

Examinons tout d'abord la taille des messages. Les champs z et s sont tels que $z \cap s = \varnothing$; de plus z et s sont inclus dans l'ensemble des identités des processus. La complexité spatiale des messages est donc $n.t$ où t désigne la plus grande identité des processus. En ce qui concerne le nombre de messages, chaque processus P_i distinct de P_{init} reçoit exactement un message *explorer* et émet au plus un message *rebrousser*. On a donc $n-1$ messages *explorer*, au plus $n-1$ messages *rebrousser* et $n-1$ messages *terminé*. Le nombre total de messages est donc compris entre $2(n-1)$ et $3(n-1)$, où n est le nombre de processus du réseau.

3.4 Commentaires

L'algorithme 3.3 réalise, pour le compte du processus P_{init}, une exploration en profondeur du réseau ; le champ *s* de l'information de contrôle transportée par les messages joue, dans le contexte distribué, le même rôle que la pile dans le contexte séquentiel. Sa complexité en nombre de messages est linéaire. Il en est de même du temps d'exécution : si le temps d'acheminement d'un message entre deux processus quelconques est borné par Δ, il est possible de quantifier ce temps d'exécution (on considère que les traitements ont des durées nulles ; ces temps peuvent être « absorbés » dans Δ). Il est au maximum égal à $[2(n-1)+d].\Delta$, où d est le diamètre du graphe G $(1 \leq d \leq n-1)$ (dans la phase de construction un seul message *explorer* ou *rebrousser* circule à un instant donné et le message *terminé* est diffusé en parallèle).

L'algorithme 3.2 réalise, pour le compte de P_{init}, une exploration en largeur d'abord du réseau (exploration parallèle); sa complexité, en nombre de messages, peut être plus élevée que celle de l'algorithme 3.3 – ceci est dû au fait que le contrôle exercé, moins riche, ne suffit pas à éviter que des processus soient atteints par des messages *explorer* n'apportant aucune information nouvelle. Par contre, le temps d'exécution est au maximum égal à $2d_{init}\Delta$, où d_{init} est l'écartement de P_{init} (c'est-à-dire la distance maximale de P_{init} à tout autre site du réseau ; rappelons que la distance entre deux sites est la longueur minimale des chaînes les reliants).

Les deux techniques que nous avons présentées ici peuvent être utilisées pour le calcul distribué d'un extrémum et du routage associé (problème dit de « l'élection » dans un système distribué : l'élection pouvant être lancée par un ou plusieurs processus simultanément, plusieurs arborescences partielles, chacune étant affectée du poids associé à un processus donné [HMR 86b], peuvent se développer concurremment ; à la fin de l'algorithme seule l'arborescence relative au poids extrémal recouvrira le réseau).

4. Interrogation du réseau par un processus

4.1 Problème et principe de résolution

Nous examinons ici le problème suivant : un processus désire acquérir une connaissance qu'il ne possède pas initialement, et doit pour cela interroger tous les processus du réseau. Chacun des processus, lorsqu'il est interrogé, fait parvenir sa connaissance locale au processus interrogateur, qui peut ainsi calculer la connaissance cherchée à partir des informations reçues (cette connaissance est dite *statique* au sens où elle est indépendante de l'instant auquel elle est calculée).

Nous supposons qu'il existe un ensemble des connaissances K, muni d'une loi de

composition T, associative, commutative, possédant un élément neutre ε et un élément absorbant α. Concrètement, T représente l'aggrégation de deux connaissances, ε est l'absence de connaissance et α est la connaissance dominante ou encore la possibilité de conclure.

Chaque site du réseau possède initialement une connaissance locale $k_i \in K$, les k_i étant des constantes, ce qui assure que la connaissance globale recherchée K est statique. Le problème, spécifique à un processus que nous désignerons par P_{init}, est le suivant : calculer $S = k_1\ T\ k_2\ T\ \dots\ T\ k_n$.

Ce problème peut être résolu par une technique de diffusion contrôlée avec réponse, voisine de celle de §3. L'exploration peut être parallèle (en largeur) ou séquentielle (en profondeur). Dans un cas comme dans l'autre, le principe est le même : P_{init} diffuse à travers le réseau une information, qui est ici une requête (demande de connaissance). Lorsqu'un processus reçoit la requête pour la première fois, il interroge à son tour le sous-ensemble de ses voisins qui, à sa connaissance, n'ont pas encore été atteints par la requête ; lorsqu'il a reçu toutes les réponses attendues, il les synthétise (au moyen de T) et répond à l'émetteur de la requête ; dans le cas où il sait que $k_i = \alpha$, ou bien s'il n'a pas de voisins à interroger, il répond immédiatement. Toute réception ultérieure de la requête provoque le renvoi immédiat de la réponse ε à l'émetteur de celle-ci. L'algorithme est terminé dès que P_{init} a reçu toutes les réponses attendues ou bien dès qu'il peut conclure (s'il sait que $k_{init} = \alpha$). Dans les algorithmes développés ci-après, le test $k_i = \alpha$ ne rendra la réponse vrai que si la valeur de α est connue *a priori* par les processus (cela dépend des cas).

4.2 L'algorithme avec exploration parallèle

Les messages sont ici de deux types :

- *(requête,z)* où z est la connaissance de contrôle de diffusion ayant la même signification que précédemment (cf. §2 et §3)
- *(réponse,x)* où x est la connaissance (de donnée) renvoyée par un processus en réponse à la première requête reçue.

Le contexte local d'un processus comporte, outre les variables $atteint_i$ et $nbacq_i$, comme dans l'algorithme 3.2, la variable r_i (initialisée avec k_i) qui contient la connaissance locale de P_i à tout instant (aussi bien la connaissance initiale que la connaissance reçue par P_i lors du fonctionnement de l'algorithme), relative à la requête. La variable booléenne $terminé_i$, initialisée à **faux**, ne sert que pour le commentaire et la preuve de l'algorithme (variable muette). Le texte de P_i est alors :

```
lors de réception de (requête,z) depuis Pj
faire
     si atteinti alors envoyer (réponse,ε) à Pj
     sinon
          atteinti:=vrai;
          si ri=α alors envoyer (réponse,α) à Pj (*terminéi=vrai*)
          sinon
               soit y=voisinsi-z;
               cas
                    y=∅ → envoyer (réponse,ri) à Pj (*terminéi=vrai*)
                    y≠∅ → predi:=j;
                    nbrepi:=card(y);
                    ∀l∈y : envoyer (requête,z∪y) à Pl;
               fcas
          fsi
     fsi
fait

lors de réception de (réponse,x) depuis Pj
faire
     si ri≠α alors
          (* Pi sait que ri≠α ou Pi ne connait pas la valeur α *)
          ri:=ri T x;
          nbrepi:=nbrepi-1;
          si (nbrepi=0 ou ri=α) alors
               cas
                    i≠init → envoyer (réponse,ri) à Ppredi
                              (* terminé = vrai *)
                    i=init → (* ri contient la connaissance recherchée *)
          fsi
          (* sinon la réponse α a déjà été envoyée lorsque ri a pris la *)
          (*  valeur α ; on ignore alors ce message *)
     fsi
fait
```

P_{init} lance l'algorithme en simulant la réception d'un message (*requête*,{*init*}) depuis P_{init}. L'algorithme est terminé dès que $nbrep_{init}$ devient égal à *0* ou r_{init} devient égal à α.

4.3 Principe de la preuve

Notation :

Pour tout processus P_i nous désignons par :

- $r_i^{(0)}$ la connaissance initiale de P_i, c'est-à-dire k_i,
- $r_i^{(f)}$ la connaissance finale de P_i, c'est-à-dire la valeur de r_i lorsque la variable auxiliaire $terminé_i$ passe à **vrai** (pour $i \neq init$, c'est la valeur de k_i renvoyée en réponse au premier voisin qui l'a interrogé),

- $y_i = voisins_i - z$, calculé éventuellement lorsque $atteint_i$ devient **vrai** (si y_i n'est pas calculé, nous prendrons par convention $y_i = \varnothing$).

On peut alors facilement montrer, par récurrence sur les chemins de l'arborescence de contrôle établie par la diffusion contrôlée des messages *requête*, que :

$$(1) \quad \forall P_i \in X : r_i^{(f)} = \begin{cases} r_i^{(0)} \; T \; (\underset{j \in y_i}{T} \; r_j^{(f)}) & \text{si } y_i \neq \varnothing \\ r_i^{(0)} & \text{si } y_i = \varnothing \end{cases}$$

On en déduit, en répercutant cette relation, à partir de P_{init}, de proche en proche sur les voisinages et en tenant compte du fait que α est un élément absorbant :

$$(2) \quad r_{init}^{(f)} = r_{init}^{(0)} T \; (\underset{j \in \cup_{i \in X} y_i}{T} \; r_j^{(0)})$$

Or, on a la propriété suivante :

$$(3) \quad (\forall l \in X : r_l^{(0)} \neq \alpha) \quad \Rightarrow \quad X \subset \bigcup_{i \in X} y_i$$

L'hypothèse se place dans le cas où la diffusion de la requête couvrira tout le réseau. Soit alors $P_j \in X$, et P_l le premier processus dont P_j reçoit un message *requête* ; on a donc, par construction : $j \in y_l$, ce qui démontre la propriété. La preuve de l'algorithme est alors obtenue en considérant deux cas :

a) $\quad \exists i \in X : r_i^{(0)} = \alpha$

Alors (1) et (2) impliquent :

$$r_{init}^{(f)} = \alpha = S$$

b) $\quad \forall i \in X : r_i^{(0)} \neq \alpha$

Alors (2) et (3) impliquent :

$$r_{init}^{(f)} = r_{init}^{(0)} \; T \; (\underset{j \in X}{T} \; r_j^{(0)}) = S$$

4.3 L'algorithme avec exploration séquentielle

Comme dans l'algorithme 3.3, il y a trois types de messages, mais une connaissance de donnée doit être transportée par les messages du type *rebrousser* ou *terminé*, qui assurent la « remontée » des réponses vers la racine P_{init}. Ces messages se présentent donc sous la forme suivante :

- *(requête,z,s)* identique au message *explorer* de l'algorithme 3.3.

• *(réponse,z,s,x)* identique au message *rebrousser* de l'algorithme **3.3**, avec le champ supplémentaire x qui transporte la connaissance renvoyée en réponse à la requête.

• *(terminé,x)* identique au message *terminé* de l'algorithme **3.3**, le champ x ayant la même signification que ci-dessus.

Le contexte local d'un processus est identique à celui de l'algorithme **3.3**, avec une variable supplémentaire r_i comme en **4.2**.

Le texte de P_i est alors le suivant :

```
lors de réception de (requête,z,s) depuis Pj
faire
      si ri=α alors envoyer (terminé,α) à Pj
      sinon
           soit y=voisinsi-z;
           cas
                y=∅ et s=∅ → envoyer (terminé,ri) à Pj
                y=∅ et s≠∅ → envoyer (rebrousser,z∪{i},s,ri) à Pj
                y≠∅ →
                     predi:=j;
                     soit l=élément(y);
                     envoyer (explorer,z∪{i},s∪y-{l}) à Pl
           fcas
      fsi
fait

lors de réception de (rebrousser,z,s,x) depuis Pj
faire
      ri:=ri T x;
      si ri=α alors envoyer (terminé,α) à Ppredi (* si i≠init *)
           sinon
                soit y=voisinsi-z;
                cas
                     y=∅ →
                          envoyer (rebrousser,z,s,ri) à Ppredi
                          (* si i≠init *)
                     y≠∅ →
                          soit  l=élément(y);
                          envoyer (explorer,z,s-{i}) à Pl
                fcas
      fsi
fait

lors de réception de (terminé,x) depuis Pj
faire
      ri:=ri T X;
      si i≠init alors envoyer (terminé,ri) à Ppredi
      fsi
fait
```

P_{init} lance l'algorithme en simulant la réception d'un message (*explorer*,{*init*},∅) depuis P_{init}. L'algorithme est terminé dès que P_{init} reçoit un message *terminé* ou dès que r_{init} devient égal à α.

4.4 Remarques

A l'issue de l'algorithme, seul le processus P_{init} est assuré de posséder la connaissance globale S :

$$S = \underset{i \in X}{T} \; k_i$$

En effet, certains processus ne collectent qu'une information partielle, et cet effet est encore renforcé par le contrôle de la diffusion de la requête. L'algorithme est donc bien adapté à la nature du problème posé, à savoir : un processus particulier cherche à acquérir une connaissance globale qu'il ne possède pas initialement. Ce processus peut ensuite diffuser aux autres processus la connaissance acquise si le problème posé le nécessite ; si tel est le cas une arborescence peut être construite durant l'interrogation et la diffusion de $r_{init}^{(f)}$ requiert alors $n-1$ messages. Les exemples ci-dessous entrent dans le cadre de ce paragraphe.

4.5 Exemples

Nous donnons quelques exemples de connaissance globale recherchée par un site, pour illustrer l'utilisation du modèle formel présenté. Seules les informations $k, T, \alpha, \varepsilon, k_i$ relatives au problème sont données ; leur insertion dans les modèles étant triviale.

4.5.1 Calcul d'un maximum

Le calcul d'un maximum est le fondement des algorithmes de tri [ZAC 85] et d'élection [PAC 84]. Chaque processus étant muni d'un poids, P_{init} désire connaître le poids de valeur maximale. Si on considère que les poids sont des entiers naturels nous avons :

$K = N$ (entiers naturels)

T = fonction *max*

$\varepsilon = 0$

α = majorant des poids si on en connaît un, ou $+\infty$ sinon

$\forall i : r_i^{(0)} = k_i$ = poids de P_i

Nous avons vu au §3 comment un tel calcul du maximum pourrait être utilisé pour réaliser une

élection lorsque les processus ont des poids différents (l'élection y inclurait en plus le calcul d'un routage). Il convient de remarquer qu'en prenant pour fonction T l'opération $+$, P_{init} calculera la somme des poids, etc... Le lecteur peut imaginer d'autres calculs distribués en fonction de K, T, ε, α.

4.5.2 Apprentissage du réseau

Nous avons vu dans les hypothèses que les processus ne connaissent ni le nombre de processus ni la structure du réseau qui les connecte. Pour un processus donné P_{init}, apprendre la topologie du réseau consiste à apprendre l'ensemble des canaux (i,j) puisque par hypothèse le réseau est connexe (pas de processus isolé). Cette connaissance est acquise par P_{init} en prenant :

$K = P(X^2)$ X désigne l'ensemble des processus et $P(X^2)$ l'ensemble des parties de X^2

$T = \cup$

$\varepsilon = \emptyset$

$\alpha = X^2$

$\forall i \in X : r_i^{(0)} = k_i = \{(i,j) \mid j \in voisins_i\}$

Le cas particulier où $k_i = \alpha$ indique que P_i connaît la structure du réseau qui est un graphe complet.

4.5.3 Appartenance à une structure particulière

Le modèle peut être utilisé pour permettre à un processus donné P_{init} de savoir s'il appartient ou non à une certaine structure du réseau, tel qu'un cycle par exemple. Considérons le cas de l'ensemble de blocage. Etant donné un graphe orienté $G=(X,\Gamma)$, un ensemble de blocage est un ensemble de sommets B inclus dans X, tels que :

i) $\Gamma(B) \subseteq B$

ii) $\forall P_i \in B : \Gamma(P_i) \neq \emptyset$

En remplaçant la notion de *voisins* par celle de *successeurs* (on est dans un réseau orienté) on se ramène au modèle. Un processus P_{init} qui désire savoir s'il appartient à une telle structure de blocage lancera une interrogation avec :

$K=\{oui, non\}$ (identifié à l'algèbre de Boole $\{vrai, faux\}$)

$T=\vee$ (opérateur logique *ou*)

$\varepsilon=oui$

$\alpha=non$

$r_i^{(0)} = k_i = (\Gamma(P_i) \neq \emptyset)$

Dans une telle interrogation l'exploration lançée par P_{init} ne touchera que les processus de $\Gamma^*(P_{init})$ (descendants de P_{init}). De plus tout processus connaît le majorant α. Il est à noter que dans une telle recherche sur l'appartenance à une structure particulière, contrairement aux deux exemples précédents, la connaissance cherchée est spécifique à P_{init}. Autrement dit *deux processus différents* peuvent obtenir *deux réponses différentes.*

En effet la connaissance cherchée fait intervenir explicitement P_{init}.

$$S = \underset{i \in \Gamma^*(P_{init})}{T} k_i$$

Une telle interrogation constitue l'une des bases d'un algorithme de détection distribuée de l'interblocage [HMR 86a]. Dans un tel algorithme un autre problème doit de plus être résolu : la structure sur laquelle est effectuée l'interrogation (graphe des attentes entre processus) peut évoluer durant l'interrogation. Il convient alors d'introduire un autre outil : le concept de période d'observation [CHA 85,WUU 85] qui, dans le cas de la recherche de propriétés stables – telles que la terminaison ou l'interblocage – permet d'assurer que le processus interrogateur P_{init} obtiendra une réponse correcte bien que les k_i puissent évoluer entre le début et la fin de l'interrogation : les k_i sont ici *dynamiques.*

5. Conclusion

Un concept général, le *contrôle des transferts de connaissance*, a été dégagé. Il est mis ici en œuvre par une technique consistant à faire transporter, par les messages de l'algorithme, une information de contrôle ; celle-ci permet à chaque processus d'exercer un contrôle local plus efficace, le but recherché étant l'élimination *a priori* – c'est-à-dire avant l'émission – des messages perçus comme inutiles par le processus.

Les résultats de complexité obtenus dans le cas de la diffusion montre que cet objectif est en général atteint, et ceci avec une augmentation de la taille des messages qui reste linéaire par rapport au nombre de processus. Il en est de même pour les deux autres modèles abordés : le parcours de réseau et l'apprentissage, par un processus donné, d'une information de nature globale.

Des deux modes de parcours mis en évidence, le second – exploration en profondeur – est particulièrement significatif : les meilleurs résultats connus jusqu'à présent pour ce mode d'exploration faisaient état d'une complexité en nombre de messages de l'ordre du nombre de lignes (donc éventuellement quadratique) et d'une complexité linéaire en temps ; notre technique permet de réduire la première à l'ordre du nombre de processus (donc linéaire) sans augmenter la seconde et en gardant une taille de message linéaire également.

Les modèles de parcours et d'apprentissage sont des fonctions de base utilisées dans de nombreux algorithmes distribués. Couplées au concept général de contrôle des transferts de connaissance, elles constituent donc un des outils fondamentaux de l'algorithmique distribuée.

Références

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

[BOU 85] BOUGE L.
Symmetry and Genericity for CSP Distributed Systems,
Rapport de recherche, LITP, Université de Paris VII, (May 1985), 22 p.

[CHA 82] CHANG E.
Echo Algorithms : Depth Parallel Operations on General Graphs,
IEEE Trans. on Soft. Eng., Vol. SE-8, n°4, (July 1982), pp. 391-401.

[CHE 83] CHEUNG T.
Graph Traversal Techniques and the Maximum Flow Problem in Distributed Computation,
IEEE Trans on Soft. Eng., Vol. SE-9, n°4, (July 1983), pp. 504-512.

[CMH 83] CHANDY K.M., MISRA J., HAAS J.
Distributed Deadlock Detection,
ACM TOCS, Vol. 1, n°2, (May 1983), pp. 144-156.

[CHA 85] CHANDY K.M., MISRA J.
Paradigm for Detecting Quiescient Properties
in Distributed Systems NATO ASI series, Vol. F13,(1985), pp. 325-341.

[DIS 80] DIJKSTRA E.W., SCHOLTEN C.S.
Termination Detection for Diffusing Computations,
Inf. Proc. Letters, Vol. 11, n°1, (August 1980), pp. 1-4.

[FRR 82] FRANCEZ N., RODEH M.
Achieving Distributed Termination without Freezing,
IEEE Trans. Soft. Eng., Vol SE-8, (1982), pp. 287-292.

[HOA 78] HOARE C.A.R.
Communicating Sequential Processes,
Comm. ACM, Vol. 21, n°8, pp. 666-677.

[HMR 86a] HELARY J.M., MADDI A., RAYNAL M.
Controlling Knowledge Transfers in Distributed Algorithms, Application to Deadlock Detection,
Rapport de recherche INRIA n°493, (Mars 1986), 32p.

[HMR 86b] HELARY J.M., MADDI A., RAYNAL M.
Calcul Distribué d'un Extremum et du Routage Associé dans un Réseau Quelconque,
Rapport de recherche INRIA n°516, (Avril 1986), 36p.

[LIN 85] LINN J., RICHARD J.
The Features and Facilities of ESTELLE,
Protocol Specification, Testing and Verification,
M.DIAZ (editor) IFIP, (1986), pp.271-296.

[LIS 83] LISKOV B., SCHEIFLER R.
Guardians and Actions : Linguistic Support for Robust, Distributed Programs
ACM TOPLAS, vol.5 ,n°3,(July 1983), pp.381-404.

[MIS 82] MISRA J., CHANDY K.M.
Termination Detection of Diffusing Computations in CSP,
ACM TOPLAS, Vol. 4, n°1, (January 1982), pp. 37-43.

[PAC 84] PACHL J.A., KORACH E., ROTEM D.
Lower Bounds for Distributed Maximum Finding Algorithms,
Journal of the ACM, vol.31, n°4, (October 1984),pp.905-918.

[RAY 85] RAYNAL M.
Algorithmes Distribués et Protocoles,
Eyrolles Ed., (Septembre 1985), 144 p.

[SEG 83] SEGAL A.
Distributed Network Protocols,
IEEE Trans. on Inf. Theory, Vol. IT-29, n°1, (January 1983), pp. 23-35.

[SCH 85] SCHNEIDER F.P.
Paradigms for Distributed Programs,
in Distributed Systems, Springer Verlag Ed., LNCS ISO, (1985), pp. 431-480.

[SIL 81] SILBERSCHATZ A.
Port Directed Communication,
Computer Journal, Vol. 24, (1981), pp. 78-82.

[WAT 83] WATSON R.W.
Identifiers (Naming) in Distributed Systems,
in Distributed Systems, an Advanced Course, Springer Verlag, (1983), pp. 191-210.

[WEG 83] WEGNER P., SMOLKA S.A.
Processes, Tasks and Monitors : a Comparative Study of Concurrent Programming Primitives,
IEEE Trans. on Soft. Eng., Vol. SE-9, n°4, (July 1983), pp. 446-462.

[ZAC 85] ZAKS S.
Optimal Distributed Algorithms for Sorting and Ranking,
IEEE Trans. on Comp. Vol. C34, n°4,(April 1985), pp.376-379.

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