Détection de la terminaison: un algorithme fondé sur une approche observationnelle

Noël Plouzeau

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° 610

DÉTECTION DE LA TERMINAISON: UN ALGORITHME FONDÉ SUR UNE APPROCHE OBSERVATIONNELLE

Noël PLOUZEAU

Février 1987

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Détection de la Terminaison : un Algorithme Fondé sur une Approche Observationnelle

Termination Detection : an Algorithm Based on Observation

Publication Interne n° 331

Décembre 1986 - 12 pages

Noël Plouzeau
IRISA
Université de Rennes 1
Campus de Beaulieu
35042 RENNES CEDEX

Résumé

Cet article présente un algorithme de détection de la terminaison d'un calcul dans un système réparti. Le calcul de détection repose sur un principe d'observation de l'activité du calcul principal dont la terminaison doit être détectée. Tous les processus de détection exécutent le même texte : il n'y a pas de processus privilégié. En ce sens, l'algorithme est symétrique. Aucun processus ne lance de calcul d'interrogation dans le réseau (à l'aide d'un jeton ou d'une vague de messages) pour apprendre l'état des processus du système. Chaque processus détecte la terminaison lorsqu'elle se produit.

Abstract

In this paper a termination detection algorithm for distributed computation is presented. The detection method is based on the observation of the main computation activity. Every detecting process executes the same code : there is no leader. In that sense, the algorithm is symmetrical. No process has to send some message wave in order to learn the state of the processes in the system. When termination occurs, every process will notice this event after some finite time.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
U.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

La détection de la terminaison d'un calcul est un exemple-type des problèmes de détection de l'occurrence de propriétés de stabilité dans un système distribué. On peut classer les différents algorithmes de détection de la terminaison connus en fonction du type du calcul de détection.

1. Un seul processus du système, que nous appelerons le détecteur, est chargé de la détection de la terminaison du calcul principal. Le principe de base de ce type d'algorithme est l'interrogation, par le détecteur, de chacun des processus afin qu'ils lui communiquent leur état. Cette interrogation emploie des messages circulant entre le détecteur et les autres processus [Dijkstra et al. 83, Chandy et Lamport 85]. Au bout d'un temps fini, le processus détecteur obtient le résultat de son calcul de l'état global du système. Ce résultat lui permet :
   - soit de déclarer que le calcul principal est terminé : il signale alors ce fait à tous les autres processus du système en diffusant un message d'information,
   - soit de constater que le résultat du calcul ne permet pas de conclure de manière certaine à la terminaison du calcul principal : le processus détecteur lance alors un nouveau calcul de détection dans le réseau.
2. Un processus du système lance un calcul pour capter l'état global du système. Pendant que ce calcul progresse dans le système, un processus constate que les résultats obtenus permettent de conclure à la terminaison du calcul principal. Il signale alors à tous les processus que le calcul principal est terminé. Dans ce type d'algorithme, le processus qui lance le calcul de détection de la terminaison et le processus qui détecte la terminaison peuvent être distincts [Misra 83].
3. Un processus particulier, unique, active les processus du système lors de l'initialisation du calcul principal, en utilisant pour ce faire un calcul diffusant d'activation. Ce même processus joue ensuite le rôle de détecteur de la terminaison. Il attend pour cela la réponse au calcul diffusant d'activation. Cette réponse arrive au détecteur si et seulement si le calcul principal est terminé. Lorsque le détecteur reçoit la réponse, il signale la fin du calcul principal à tous les processus du système en diffusant un message d'information [Dijkstra et Scholten 80].

L'algorithme présenté ici propose une quatrième façon de détecter la terminaison. L'approche originale retenue s'appuie uniquement sur l'observation de l'activité des messages de calcul dans le réseau de communication. Cette observation est effectuée par tous les processus du système. Lorsqu'un processus a détecté la terminaison du calcul principal il n'en informe pas les autres processus. Ceux-ci ont découvert seuls, ou vont découvrir, que le calcul principal est terminé. Aucun processus ne joue le rôle de détecteur unique : tous sont détecteurs.

Nous ne nous intéressons pas au cas des pannes de sites, le but de cet article étant la

présentation de la technique de l'observation distribuée, au moyen d'un algorithme simple. Nous n'emploierons pas la communication par variable partagée entre sites voisins sur un anneau [Saikkonen et Rönn 86] : la communication entre sites se fera exclusivement par messages. Contrairement à [Lai 85], la création et la suppression dynamique de sites n'est pas prise en compte.

2 Hypothèses

2.1 Propriétés requises du réseau

L'algorithme fait les hypothèses suivantes sur le réseau qui relie les différents sites du système.

- Le déséquencement, la duplication, la perte et l'altération des messages du calcul principal sont interdits.
- L'algorithme de détection n'utilise que la diffusion fiable comme primitive de communication entre les sites du système. Le déséquencement, la duplication, la perte et l'altération des messages de contrôle entre le site origine de la diffusion et un destinataire quelconque sont interdits.

2.2 Définition du système à observer

Le calcul principal dont on veut détecter la terminaison est régi par les règles suivantes :

1. Sur chaque site S_i du système distribué se trouve un processus C_i participant au calcul principal.
2. Un processus du calcul principal peut être soit dans l'état *actif* soit dans l'état *passif*.
3. Pour émettre un message de calcul un processus doit nécessairement être dans l'état *actif*.
4. Un processus dans l'état *actif* peut passer arbitrairement dans l'état *passif*.
5. Un processus dans l'état *passif* passe dans l'état *actif* si et seulement si il reçoit un message de calcul ; le changement d'état est immédiat. Il n'y a donc pas d'activation spontanée de processus de calcul.
6. Au début du calcul principal tous les processus sont dans l'état *actif*.

Définition : On dit que le calcul principal est terminé si et seulement si tous les processus sont dans l'état *passif* et il n'y a aucun message de calcul en transit dans le réseau.

Notation : Un site du système est noté S_i, où i est un élément de l'intervalle entier [1..n], n étant le nombre de sites du système. Les variables locales d'un processus appartenant à un site S_i seront toujours indicées par i.

3 Principe de l'algorithme

L'algorithme de terminaison est exécuté sur chaque site du système : comme nous l'avons précisé, il n'y a pas de site « détecteur » unique, qui prendrait seul la décision de déclarer le calcul terminé.

On trouve sur chaque site S_i du système, en plus du processus du calcul principal C_i, les processus M_i et D_i définis comme suit.

M_i Ce processus est chargé de noter l'activité de C_i au sein du calcul principal. Le processus M_i observe l'activité du processus C_i en comptant les messages de calcul émis et reçus par celui-ci. Il possède deux variables locales *calc_émis*$_i$ et *calc_reçus*$_i$, qui sont des vecteurs d'entiers indexés par l'ensemble des noms des processus. Le processus M_i obéit aux règles suivantes :

1. Lorsque C_i émet un message de calcul, quel qu'en soit le destinataire C_j, M_i augmente la valeur de l'élément *calc_émis*$_i$[j] d'une unité.
2. Lorsque C_i reçoit un message de calcul, quel qu'en soit l'expéditeur C_j, M_i augmente la valeur de l'élément de vecteur *calc_reçus*$_i$[j] d'une unité.
3. Lorsque le processus de calcul C_i devient passif, le processus M_i associé diffuse vers tous les processus D du système (y compris D_i) un message *signal_passif* qui transporte la valeur des vecteurs *calc_émis*$_i$ et *calc_reçus*$_i$. Les vecteurs *calc_émis*$_i$ et *calc_reçus*$_i$ sont ensuite remis à zéro.

D_i Ce processus est chargé de la détection de la terminaison du calcul principal. Le processus D_i détient localement deux matrices d'entiers, *total_émis*$_i$ et *total_reçus*$_i$. Le comportement de D_i est définie par les règles suivantes :

1. L'élément *total_émis*$_i$[j][k] contient à tout instant le nombre des messages de calcul émis par C_j vers C_k, tel que le connaît D_i. Cette connaissance est mise à jour lors de la réception par D_i d'un message *signal_passif* émis par M_j : D_i ajoute à *total_émis*$_i$[j] la valeur du vecteur *calc_émis*$_j$ envoyé par M_j dans son message *signal_passif*.
2. L'élément *total_reçus*$_i$[j][k] contient à tout instant le nombre des messages de calcul reçus par C_i en provenance de C_k, tel que le connaît D_i. Cette connaissance est mise à jour lors de la réception par D_i d'un message *signal_passif* émis par M_j : D_i ajoute à *total_reçus*$_i$[j] le vecteur *calc_reçus*$_j$ envoyé par M_j dans son message *signal_passif*.

3. Lorsque le processus D_i constate que pour tout j et tout k on a $total\_émis_j[j][k] = total\_reçus_j[k][j]$, et qu'il a reçu au moins un message *signal_passif* de chacun des processus du système alors D_i déclare le calcul principal terminé.

4 Texte de l'algorithme

L'algorithme est donné par un ensemble de procédures qui sont appelées lors de l'occurrence d'un événement dans un processus. Ces procédures sont exécutées de manière atomique.

Texte du processus C_i

```
   calc_reçus_i : tableau [1..n] de naturel init 0;
   calc_émis_i  : tableau [1..n] de naturel init 0;

   co
      L'élément de tableau calc_reçus_i[j] contient le nombre
      des messages de calcul reçus par P_i, en provenance de P_j,
      depuis le dernier passage de C_i dans l'état actif.

      L'élément de tableau calc_émis_i[j] contient le nombre
      des messages de calcul envoyés par P_i à P_j depuis le
      dernier passage de C_i dans l'état actif.
   fco
proc émission_calcul(j);
   co
      Cette procédure est appelée lorsque C_i envoie un message
      de calcul à C_j.
   fco

   calc_émis_i[j]:=calc_émis_i[j]+1;
fproc
proc réception_calcul(j)
   co
      Cette procédure est appelée lorsque C_i reçoit un message de
      calcul venant de C_j.
   fco

   calc_reçus_i[j]:=calc_reçus_i[j]+1;
fproc
```

```
proc devenir_passif;
   co
      Cette procédure est appelée lorsque le processus C_i passe dans
      l'état passif.
      L'opération de diffusion utilisée ci-dessous envoie
      le message à diffuser à tous les processus D_x du système,
      y compris à D_i lui-même.
   fco

   diffuser signal_passif(i,calc_émis_i,calc_reçus_i);
   calc_émis_i:=0;
   calc_reçus_i:=0;
fproc
```

Texte du processus D_i

```
var
   total_reçus_i : tableau [1..n][1..n] de naturel init 0;
   total_émis_i : tableau [1..n][1..n] de naturel init 0;

   co
      L'élément de tableau total_reçus_i[j][k] contient le
      nombre des messages de calcul reçus par C_j
      en provenance de C_k, tel que le connaît D_i.
      Cet élément est mis à jour lors de la réception d'un message
      signal_passif diffusé par M_j lorsque C_j passe dans l'état
      passif.

      L'élément de tableau total_émis_i[j][k] contient le nombre
      des messages de calcul envoyés par C_j
      à C_k, tel que le connaît D_i.
      Cet élément est mis à jour lorsque D_i reçoit un message
      signal_passif émis par M_j.
   fco

   participants_i : tableau [1..n] de booléen init faux;

   co
      L'élément participants_i[j] vaut vrai si D_i a reçu au
      moins un message signal_passif venant de M_j.

      A l'initialisation du système, tous les processus sont dans
      l'état actif.
   fco
```

```
proc réception_passif(j,copie_calc_émis,copie_calc_reçus);
   co
      Cette procédure est appelée lors de la réception d'un message
      signal_passif par P_i. Ce message provient de P_j, les
      champs copie_calc_émis et copie_calc_reçus correspondent aux champs
      calc_émis et calc_reçus diffusés par la
      procédure devenir_passif.
   fco

   total_émis_i[j]:=total_émis_i[j]+copie_calc_émis;
   total_reçus_i[j]:=total_reçus_i[j]+copie_calc_reçus;
   participants_i[j]:=vrai;
   si ∀j, ∀k, total_émis_i[j][k]=total_reçus_i[k][j] et participant_i[j]
         alors co Le calcul principal est déclaré terminé fco
   fsi;
fproc
```

5 Preuve de bon fonctionnement de l'algorithme

Notations

La preuve s'appuie sur l'existence d'un référentiel de temps global ; ce temps est abstrait : il n'est basé que sur le fait que l'émission d'un message précède sa réception et que dans un même processus deux événements ne peuvent avoir lieu simultanément.

Supposons que D_i, processus détecteur quelconque du système, déclare le calcul terminé. Nous définissons alors les notations suivantes :

dt_i	date à laquelle D_i déclare le calcul principal terminé,
$drp_i(j)$	valeur maximale des dates de réception par D_i des messages *signal_passif* émis par M_j, et reçus avant dt_i,
dep_j	date d'émission par M_j du message reçu à la date $drp_i(j)$ par D_i,
$dec_k(j)$	date d'émission d'un message de calcul quelconque émis par C_k vers C_j,
$drc_j(k)$	date de réception par C_j du message de calcul émis par C_k à la date $drc_j(k)$.

5.1 L'algorithme ne fait pas de fausse détection

Lemme

Si D_i détecte la terminaison à la date dt_i alors :

$$dec_k(j) < dep_k \Rightarrow drc_j(k) < dep_j$$

Ce lemme signifie que tout message de calcul émis par C_k vers C_j avant la « dernière » diffusion de *signal_passif* par M_k arrive nécessairement avant la dernière diffusion de *signal_passif* par M_j. Un exemple d'exécution satisfaisant ce lemme est représenté à la figure 1, où la droite repérée par S_j porte la trace des événements qui ont lieu sur le site S_j. Les flèches représentent la transmission des messages entre les différents sites.

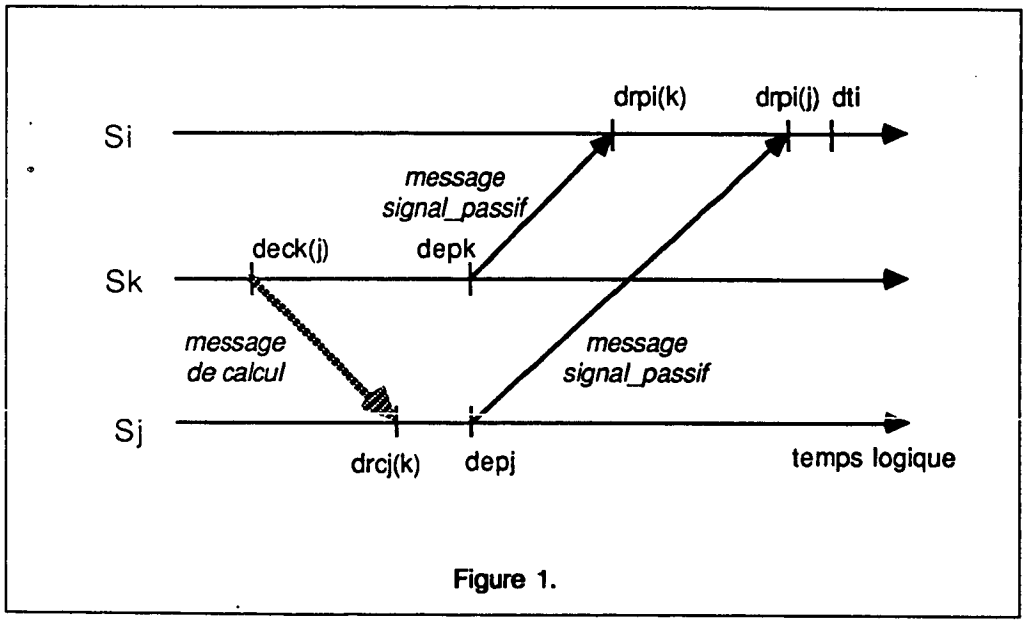

Figure 1.

Preuve du lemme

Nous allons prouver le lemme en raisonnant par contradiction. Nous prendrons comme hypothèse de travail :

$$dec_k(j) < dep_k \wedge drc_j(k) > dep_j \qquad (H)$$

On notera m_j le message de calcul émis à $dec_k(j)$ et reçu après dep_j. La figure 2 représente un exemple d'exécution satisfaisant l'hypothèse H. Puisque m_j a été émis à la date $dec_k(j)$, il est pris en compte par le champ *copie_calc_émis* d'un message *signal_passif* émis par M_k à une date

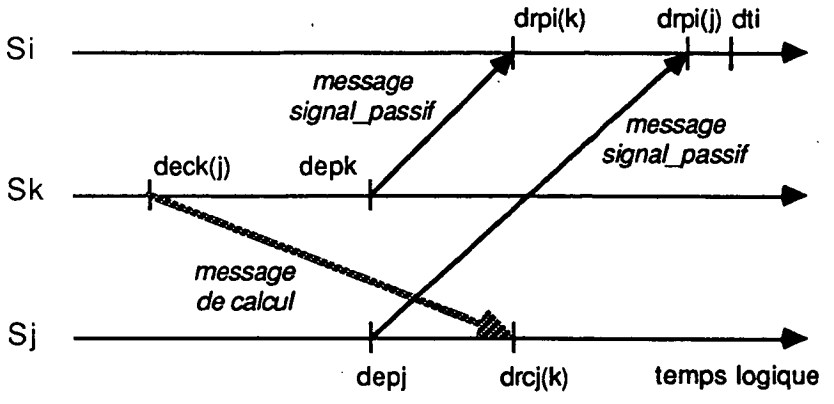

Figure 2.

inférieure ou égale à dep_k ; comme les messages de contrôle *signal_passif* ne peuvent pas se doubler en circulant de M_k à D_i, le message m_j est pris en compte dans $total\_émis_i[k][j]$ à la date dt_i. Puisque m_j a été reçu après dep_j, que les messages de contrôle ne peuvent se doubler entre M_j et D_i, m_j ne peut être pris en compte dans $total\_reçu_i[j][k]$ à la date dt_i. De plus, les messages de calcul ne peuvent pas se doubler entre C_k et C_j, donc aucun message émis par C_k après $dec_k(j)$ ne peut « simuler » la réception de m_j avant dep_j en étant pris en compte à la place de m_j. A la date dt_i, l'élément $total\_reçu_i[j][k]$ est inférieur au nombre de messages émis ; on a alors $total\_émis_k[k][j] > total\_reçu_i[j][k]$. Or nous avons fait l'hypothèse que D_i détecte la terminaison à cette date, ce qui implique l'égalité des tableaux $total\_émis_j$ et $total\_reçus_i$. Il y a contradiction : l'hypothèse **H** est donc fausse, ce qui démontre le lemme.

Théorème 1

L'algorithme ne fait pas de fausse détection : si un processus quelconque du système déclare le calcul principal terminé alors tous les processus du système sont dans l'état *passif* et il n'y a aucun message de calcul en transit dans le réseau.

Preuve du théorème 1

Nous commencerons par prouver qu'il ne saurait y avoir de processus dans l'état *actif* après qu'un processus D_i (choisi arbitrairement) ait déclaré le calcul terminé.

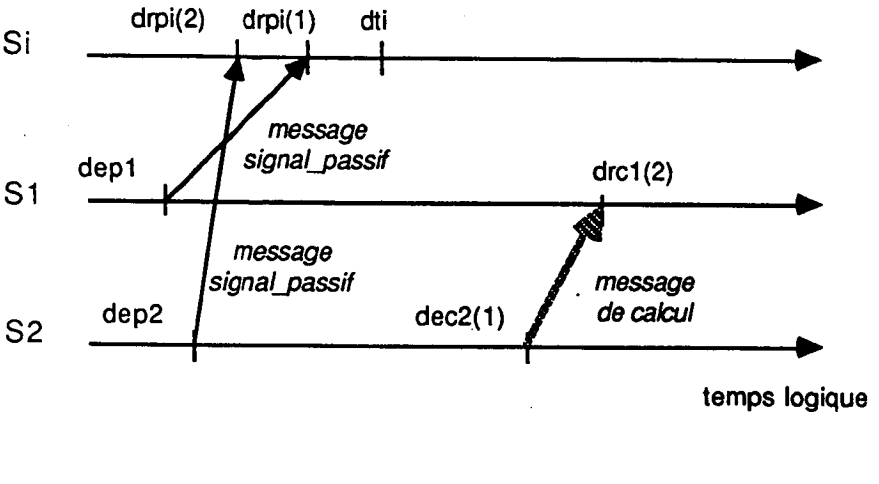

Figure 3.

Supposons qu'il existe un processus C_1 actif après dt_i (figure 3). C_1 a dû passer au moins une fois dans l'état passif avant dt_i, car le prédicat de détection de la terminaison impose la réception par D_i d'au moins un message *signal_passif* émis par M_1. Un processus ne peut passer de l'état passif à l'état actif que par réception d'un message de calcul. Etant passif à l'instant dep_1, C_1 a dû recevoir au moins un message de calcul après cette date. Soit m_1 le premier des messages de calcul reçus par C_1 après dep_1. La date de réception de m_1 sera notée $drc_1(2)$. L'émetteur de ce message sera noté C_2. En vertu du lemme, et en considérant la contraposée de l'implication figurant dans ce lemme, il apparaît que le message m_1 a été émis nécessairement **après** dep_2. Ceci implique que C_2 devait être dans l'état actif après dep_2. Cela n'est possible que si C_2 a reçu un message de calcul à la date $drc_2(3)$, postérieure à dep_2. De plus, la date $drc_2(3)$ est nécessairement antérieure à la date d'émission de m_1 et donc antérieure à $drc_1(2)$ (la comparaison de ces deux dates a un sens, bien qu'elles se rapportent à deux sites distincts, car elles sont liées par une relation de cause à effet, en l'occurrence la transmission du message m_1).

Cette situation se résume de la manière suivante : s'il existe un processus C_1, tel que C_1 soit repassé dans l'état *actif* à la date $drc_1(2)$ postérieure à dep_1, alors il existe nécessairement un autre processus C_2 distinct de C_1, qui est passé dans l'état *actif* à une date $drc_2(3)$ antérieure à $drc_1(2)$.

L'ensemble des processus du système étant statique et fini (de cardinal n), il existe nécessairement une chaîne de processus $C_1,...,C_n$, où C_{i+1} réactive C_i. Cette chaîne est acyclique en raison de la contrainte $drc_{i+1}(i+2) < drc_i(i+1)$. Le processus C_n n'ayant pas de successeur, il ne peut passer dans l'état actif après dep_n. Il ne peut donc pas activer C_{n-1}. De proche en proche, C_{i+1} ne peut activer C_i. Donc C_1 ne peut être dans l'état actif, ce qui contredit l'hypothèse de travail. Il n'existe donc pas de processus dans l'état *actif* après que D_i ait déclaré le calcul principal terminé.

De plus, il ne peut pas exister de message de calcul en transit dans le réseau après la date dt_i car si tel était le cas alors au bout d'un temps fini un processus du système recevrait nécessairement ce message et passerait dans l'état *actif*. Or nous venons de voir qu'aucun processus ne peut passer dans cet état après dt_i. Il ne peut donc y avoir de messages en transit dans le réseau après que D_i ait déclaré le calcul principal terminé.

5.2 L'algorithme détecte la terminaison au bout d'un temps fini

Théorème 2

Si le calcul principal se termine alors tout processus du système détecte cette terminaison au bout d'un temps fini.

Preuve du théorème 2

On suppose qu'à l'instant T le calcul principal est terminé : le dernier processus encore actif devient passif, il n'y a plus de message de calcul en transit dans le réseau. A l'instant T, tous les messages émis ont été reçus, et chaque site a émis vers tous les processus détecteurs la dernière partie du compte exact des messages de calcul émis et reçus. Au bout d'un temps fini, tout processus détecteur connaît pour tout j et tout k le nombre total de messages de calcul émis par C_j vers C_k et le nombre total des messages de calcul reçus par C_k en provenance de C_j ; ces deux totaux sont égaux. On a donc au bout d'un temps fini l'égalité des deux tableaux. Ceci rend vrai le premier des deux termes de la conjonction servant de prédicat de détection de la terminaison. Le second terme est également vrai car chaque processus a envoyé au moins un message *signal_passif* à l'instant T, puisque par hypothèse tous les processus sont dans l'état *actif* au début du calcul. Donc tout processus du système verra son prédicat de terminaison devenir vrai au bout d'un temps fini après la terminaison du calcul principal.

6 Conclusion

L'approche retenue pour le calcul de détection de la terminaison s'appuie uniquement sur le concept de l'observation d'une activité distribuée. L'algorithme est une illustration simple de l'emploi de cette technique. Celle-ci semble utilisable pour résoudre le problème plus général de la détection de l'occurrence de propriétés stables dans un système distribué. Ceci constitue actuellement un des sujets de recherche de l'équipe.

7 Remerciements

Cet algorithme a été conçu au sein de l'équipe *Algorithmes distribués et protocoles* à l'IRISA, dans le cadre du pôle *Algorithmes distribués* du GRECO C^3. Jean-Michel Hélary, Claude Jard et Michel Raynal ont apporté de nombreuses critiques et corrections au texte et à la preuve de cet algorithme.

8 Références

[Chandy et Lamport 85] CHANDY, K.M. et LAMPORT, L., *Distributed snapshots : determining global states of distributed systems*, ACM Trans. Comp. Syst., Vol. 3, n° 1, 1985, pp. 63-75.

[Dijkstra et Scholten 80] DIJKSTRA, E.W. et SCHOLTEN, C.S., *Termination detection for diffusing computation*, Inform. Process. Letters, Vol. 11, n° 1,1980, pp. 1-4.

[Dijkstra et al. 83] DIJKSTRA, E.W., FEIJEN, W.H.J. et VAN GASTEREN, A.J.M., *Derivation of a termination detection algorithm for distributed computation*, Inform. Process. Letters, n° 16, 1983, pp. 217-219.

[Lai 85] LAI, T.H., *Termination detection for dynamic distributed systems with non-first-in-first-out communication*, Dept. of Computer and Information, Ohio State Univ., Columbus, Ohio, 1985.

[Misra 83] MISRA, J., *Detecting termination of distributed computations using markers*, Proc. Second Annual ACM Symp. on Principles of Distributed Computing, Montreal, Août 1983, pp. 290-294.

[Saikkonen et Rönn 86] SAIKKONEN, H. et RONN, S., *Distributed termination on a ring*, BIT, n° 26, 1986, pp. 188-194.

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