Algorithme d'Arrêt Global Consistant de Systèmes Distribués à Temps de Latence Borné. (*)

Pierre MOUKELI

Institut Africain d'Informatique
(IAI) B.P. 2263 Libreville
GABON

Tel. (241) 72 00 05 Fax. (241) 72 00 11

Résumé. La programmation parallèle nécessite des outils permettant l'arrêt global des systèmes distribués. Plusieurs travaux ont été menés permettant de construire de tels outils. Cependant le nombre de messages générés par les algorithmes qui en résultent les rend inefficaces. Le présent travail qui est une amélioration d'un résultat antérieur propose un algorithme permettant un arrêt consistant du système distribué et générant un nombre de massages borné et indépendant du nombre de processus dans le système.

Mots clés: Systèmes distribués, parallélisme, point d'arrêt globaux.

Summary. Parallel programming usually require tools for halting distributed systems. Several Work had been carried out in order to build such tools. However, the number of messages generated by these algorithms makes them inefficient. The current work which improve a previous one, proposes an algorithm for consistent global breakpoints. It generates a number of message bounded and free from the process number.

Key words. Distributed systems, parallelism, breakpoints.

I. Introduction.

L'arrêt global de systèmes distribués est une opération dont la difficulté et la complexité croissent avec nombre de processus et de processeurs de la machine parallèle. Même quand un mécanisme centralisateur existe (e.g. processeur de contrôle), la décision de suspendre l'exécution des processus est souvent prise de manière concurrente par des processus s'exécutant sur des noeuds différents. L'utilisation d'algorithmes efficaces pour l'arrêt global des processus est donc indispensable, en s'appuyant éventuellement sur des mécanismes matériels facilitant la communication, la synchronisation et l'ordonnancement des processus.

Les travaux de Chandy-Lamport [Cha] et ceux de Mattern [Mat] ont permis de développer deux classe d'algorithmes [Bou] [Hab] [Nai]. Les inconvénients majeurs de ces algorithmes sont décrits dans [Mou]. Le présent article décrit une amélioration de l'algorithme décrit dans [Mou] qui présente un certains nombre d'inconvénients. Le premier est qu'un point d'arrêt pouvait démarrer sans que le point d'arrêt précédent ne soit effectivement terminé. Le deuxième inconvénient est que la consistance ne pouvait être garantie que si les événements collectés étaient estampillés, ce qui nécessitait une horloge globale. Le présent algorithme, lève ces problèmes tout en gardant le nombre de messages échangés pour un point d'arrêt indépendant du nombre de processus en cours d'exécution.

(*) Ce travail est effectué dans le cadre du projet ParCC (Parallel Computing in Central-Africa) financé par la Commission Européenne. Référence ITDC'94, N° 232-82173.

Le travail exposé s'applique principalement aux architectures dont le temps de latence des communications est borné. Le temps de latence est le temps qui s'écoule entre le moment où le premier processus engagé dans une communication est remis dans la file et le moment où le deuxième processus sera aussi remis dans la file. Cette propriété permet en particulier de garantir qu'une communication engagée se termine toujours dès lors que les partenaires sont prêts.

La deuxième partie de cet article présente quelques généralités sur les points d'arrêt. Elle définit la notion d'arrêt consistent, présente l'intérêt du présent travail et expose les hypothèses sur l'environnement cible pouvant supporter les résultats de ce travail. La troisième partie décrit les algorithmes mettant en oeuvre la procédure d'arrêt global. La preuve de la consistance de l'arrêt est établie ainsi qu'une évaluation du coût de la procédure en terme de messages échangés. La conclusion aborde les problèmes de tolérance aux pannes. Les algorithmes sont écrits dans un langage C parallèle, cependant la connaissance de ce langage n'est pas un préalable à la compréhension du travail exposé.

II. Généralités sur les points d'arrêts.

II.1. DÉFINITIONS.

Dans cet article, un système de processus parallèles (ou simplement un système) est un ensemble de processus distribués sur plusieurs processeurs, communicant par des canaux, et concourant à la résolution d'un même problème. Un processus est une ligne de contrôle pouvant être décrite par l'ensemble des états par lesquels elle passe ou l'ensemble des événements qui la font passer d'un état à l'autre.

L'état d'un processus se reflète à travers son contexte (i.e. variables, canaux, registres). Un événement est une action atomique modifiant l'état (le contexte) d'un processus, ou la composition d'actions entre elles.

L'état global d'un système est constitué de l'union des états des processus qui le composent [Cha]. Comme cet état global ne peut pas être obtenu instantanément, une procédure d'arrêt du système appelée point d'arrêt global doit être mise en oeuvre. Cette procédure est au moins partiellement basée sur des algorithmes système dont l'efficacité dépend à la fois du matériel et de la complexité en terme de messages échangés.

Une condition globale d'arrêt est une condition qui une fois vérifiée permet de décider de déclencher l'arrêt global. Elle est souvent exprimée par un prédicat qui lui même est une composition de conditions locales à plusieurs processus repartis sur des noeuds différents. Une technique de construction de tels prédicats est décrite dans [Mil] [Mou].

Un point d'arrêt global est constitué de quatre phases. La première phase consiste à évaluer la condition d'arrêt global. La deuxième phase consiste à demander l'arrêt global quand une condition globale a été validée. Cette phase peut être déclenchée concurremment par des processus repartis sur différents noeuds. Elle peut donc engendrer un flot de messages proportionnel au nombre de processus actifs. Le point d'arrêt étant actif, la troisième phase consiste à exécuter l'action justifiant l'arrêt global. Nous dirons que cette action est distribuée si à chaque processus arrêté est associée une action spécifique; sinon l'action est dite globale. Enfin, la quatrième phase consiste terminer le point d'arrêt et à réveiller les processus qui ont été arrêtés.

Dans cet article nous nous intéressons uniquement aux trois dernières phases que nous désignerons comme étant la procédure d'arrêt global.

II.2. ARRÊT GLOBAL CONSISTANT.

Un arrêt global présentera d'autant plus d'intérêt qu'il permet de vérifier certaines propriétés sur les processus impliqués par l'arrêt, en particulier les propriétés stables comme les boucles infinies, les interblocages, les terminaisons de communications, ou encore les dépendances de communication [Bou] [Nai]. Une condition générale permettant de vérifier de telles propriétés est la consistance de l'arrêt du point de vue des états et des événements. Nous allons rappeler ces notions.

Consistance de l'état global.

Propriété 1. Un arrêt global est dit consistant au sens des états si l'état obtenu pour chaque processus arrêté est équivalent à l'état que ce processus aurait collecté avant de s'arrêter [Mil].

Consistance de l'ensemble des événements.

Une coupe est une partition de l'ensemble des événements d'une exécution en deux sous-ensembles PASSE et FUTUR; la coupe étant définie par son sous-ensemble PASSE [Mat]. Une coupe C est dite consistante [Bou] [Mat] si, $\mathcal{E}$ étant l'ensemble des événements d'une exécution :

$$\forall e \in C, \forall e' \in \mathcal{E},\ e' \rightarrow e \Rightarrow e' \in C;$$ où $\rightarrow$ désigne la relation de précédence causale.

Un corollaire à cette définition établi dans [Ada] montre que cette définition revient à dire qu'à toute réception d'un message prise en compte dans la coupe, correspond une émission du même message elle aussi prise en compte dans la coupe.

Propriété 2. Un arrêt global est dit consistant au sens des événements si l'ensemble des événements qu'il a permis de collecter forme une coupe consistante [Mat].

Arrêt global consistant.

Propriété 3. Nous dirons qu'un point d'arrêt global est consistant si l'état global obtenu est consistant, et si l'ensemble des événements qu'il a permis de collecter forme une coupe consistante. C'est à dire qu'il vérifie les propriétés 1 et 2.

II.3. INTÉRÊT DU PRÉSENT TRAVAIL.

L'arrêt global des processus est souvent indispensable en phase de mise au point pour détecter des anomalies [Bur] [Jon], ou en phase d'exploitation pour prévenir un comportement aberrant du système si certains événements ou certaines conditions se produisent (e.g. cas des exceptions) [Bem] [Str]. Comme le système ne peut pas être arrêté instantanément [Cha], l'efficacité d'un point dépend de la durée qui s'écoule entre le moment où la décision d'arrêter le système est prise et le moment où le dernier processus est arrêté. Cette durée est tributaire de la complexité de l'algorithme en terme du nombre de messages échangés pour amener tous les processus à s'arrêter.

C'est sur cette complexité que repose le travail exposé dans cet article. En effet, la complexité du présent algorithme est indépendante du nombre de processus dans le système, contrairement aux algorithmes de Chandy-Lamport et de Mattern et leurs dérivés [Bou] [Cha] [Hab] [Mat] [Mil] [Nai]. De plus, l'arrêt global obtenu par le présent algorithme est consistant. Il ne nécessite pas d'estampiller les messages comme l'essentiel de tous les autres algorithmes.

II.4. HYPOTHÈSES SUR L'ENVIRONNEMENT DE TRAVAIL.

Dans le présent article nous faisons les hypothèses suivantes. Les processus parallèles communiquent par l'envoi de messages à travers des canaux binaires synchrones. Une communication se termine quand l'un au moins des deux processus impliqués est remis dans la file d'ordonnancement. Chaque processeur dispose d'un mécanisme primaire d'ordonnancement [Fei] [Str]. Quand une communication se termine, c'est ce mécanisme qui assure l'insertion du processus dans la file d'ordonnancement [INM] (e.g. gestionnaire de liens dans les machines à base de transputers). Nous supposons également que le temps de latence ne dépend que du mécanisme matériel cité ci-dessus, et est borné par une constante.

Les processus qui s'exécutent sur le même processeur partagent une zone mémoire commune accessible éventuellement en exclusion mutuelle. Le système dispose d'un mécanisme permettant de diffuser des messages, et d'effectuer des communications asynchrones pour des messages de service. A la suite d'une temporisation (*timer*) un processus est mis en fin de file d'attente. Enfin, les messages émis par un processus vers un autre sont reçus dans l'ordre d'émission, et le communications sont synchrones (sauf précision).

III. Procédure d'arrêt consistant des processus.

III.1. DESCRIPTION.

Nous supposons dans cette partie, qu'une condition globale d'arrêt a été évaluée et que la décision d'arrêter le système est prise. Des algorithmes permettant d'évaluer de telles conditions sont décrits dans [Mil] [Mou]. La condition d'arrêt peut être évaluée par plusieurs processus en même temps sur des noeuds différents. Malgré cette concurrence, un seul de ces processus doit être responsable de l'arrêt global. Un processus qui a échoué dans sa tentative de déclencher d'arrêt global peut éventuellement faire une nouvelle tentative quand le point d'arrêt global est terminé. Tout processus qui tente un arrêt global reçoit une réponse indiquant si sa tentative a été fructueuse ou non.

Principe de la procédure.

La procédure générale du point d'arrêt est basée sur l'arrêt volontaire et sur l'existence sur chaque noeud d'un processus gestionnaire du point d'arrêt appelé *manager*. L'un des managers est élu pour être le *superviseur* du point d'arrêt. Le principe est que lorsqu'une condition globale d'arrêt est vérifiée, tout processus désirant demander l'arrêt global du système exécute la fonction d'arrêt global pour envoyer un message au manager du point d'arrêt de son noeud. Le *demandeur* attend un acquittement du manager.

Le manager en recevant le message le transmet au superviseur, qui peut ainsi recevoir jusqu'à N messages (où N indique le nombre de noeuds). Le superviseur choisit le premier de ces messages, le diffuse à tous les managers et annule les autres demandes. Chaque manager en recevant le message diffusé positionne un flag visible par tous les processus du noeud et acquitte le demandeur du point d'arrêt sur ce noeud. Dès que le flag est positionné, *le point d'arrêt* est dit *actif*. Tout processus qui détecte ce flag peut décider de s'arrêter en exécutant la fonction d'arrêt individuel; un tel processus sera appelé *adhérant* du point d'arrêt.

L'acquittement mentionné plus haut est dit *positif* pour un demandeur si le message qui a été diffusé porte la signature de ce demandeur, sinon il est dit *négatif*. Plusieurs processus peuvent être demandeurs, mais le système garantit qu'un seul recevra un acquittement positif. Le processus recevant

cet acquittement positif est le *responsable* du point d'arrêt. C'est ce responsable qui décidera de réveiller le système compte tenu des conditions ayant justifié l'arrêt.

Lorsque les adhérants sont arrêtés, une action est exécutée pour collecter des observations sur chacun d'eux. Afin d'adapter l'action à chaque processus, l'algorithme d'adhésion permet à chaque adhérant de spécifier une action (i.e. fonction) qui sera exécutée pour collecter son état pendant l'arrêt. L'ensemble de ces actions est appelée *action distribuée.*

Quand le responsable décide de terminer l'arrêt, il envoie un message au superviseur qui le diffuse, signifiant la terminaison du point d'arrêt aux managers. Chaque manager positionne alors le flag de terminaison et attend éventuellement que l'action distribuée se termine sur son noeud. Ensuite il acquitte le superviseur de la terminaison de l'action distribuée, et attend un dernier message diffusé par le superviseur certifiant la fin générale de l'action distribuée et donc du point d'arrêt sur tous les noeuds.

III.2. ALGORITHMES.

La procédure d'arrêt est composée des quatre algorithmes suivants :

Arrêt global (G) : exécuté par tout processus qui veut déclencher l'arrêt global du système
Arrêt individuel (I) : exécuté par chaque processus qui veut s'arrêter quand le point d'arrêt est actif.
Manager (M) : exécuté par le processus manager sur chaque noeud, et par le processus superviseur.
Terminaison (T) : exécuté par le responsable du point d'arrêt pour terminer celui-ci.

Ces algorithmes sont exécutés en mode privilégié pour assurer l'exclusion mutuelle. Les processus ne peuvent être suspendus dans ces algorithmes que dans les points de communications ou de réception de signaux [Fei]. Le point d'arrêt comporte sur chaque noeud une structure de données notée BP dans la suite, accessible en exclusion mutuelle par les processus du noeud. Elle contient entre autres les champs suivants:

status: état du point d'arrêt (FREE : aucun point d'arrêt en cours; BUSY: demande de point d'arrêt en cours; ACTIVE: point d'arrêt actif, END: point d'arrêt en phase de terminaison)

list: File des processus arrêtés.

ctr: Nombre de processus exécutant l'action distribuée.

III.2.1. ALGORITHME G DE DEMANDE D'ARRÊT GLOBAL.

La demande d'arrêt global n'est acceptée que lorsque le prédicat global conditionnant l'arrêt est valide et qu'un point d'arrêt n'est pas en cours. Si cette condition est vraie, le processus constitue une requête d'arrêt global qu'il transmet au manager. Celui-ci lui renvoie un signal d'acquittement. Le processus vérifie que sa requête a été retenue si le numéro du processus responsable du point d'arrêt est le sien.

```
1.  int Enable_Breakpoint (Predicate p) {                    /* executed in protected mode */
2.          MESSAGE          msg ;
3.          if (Test_Predicate(p) && (BP.status == FREE)) {
4.                  BP.status = BUSY ;
5.                  Send(BP.Enable_Req) ;                    /* to the manager */
6.                  Wait_Manager_Ack () ;                    /* from the manager */
7.                  if (my_pid() == BP.pid)                  /* Request accepted */
8.                          return (1) ;                     /* Succesful request */
9.                  }
10.         return (0) ;
            }
```

III.2.2. ALGORITHME I D'ARRÊT CONDITIONNEL INDIVIDUEL.

La fonction relative à cet algorithme peut être exécutée par tout processus qui veut s'arrêter. L'arrêt n'est cependant effectif que si le point d'arrêt est actif. Dans ce cas, le processus exécute sa propre action. Si le processus est le dernier à terminer son action après que le manager ait annoncé la fin du point d'arrêt, alors le processus réveille le manager qui s'est mis en attente de ce retardataire. L'algorithme se termine en suspendant le processus pour le mettre en attente d'être réveillé par le manager à la fin du point d'arrêt.

```
        /* In this param1 refers to the first parameter of the action function, and TYPE1 is its type. (...)
           refers to the other parameters*/
1.  Individual_breakpoint (VOID (*action), TYPE1 param1, ...) {
2.          if (BP.status != ACTIVE)
3.                  return ;
4.          BP.ctr++ ;
5.          if (action != NULL)
6.                  action(param1, /* ...other param. */) ;
7.          else
8.                  default_action () ;
9.          BP.ctr-- ;
10.         if ((BP.ctr == 0) && (BP. status == END))
11.                 Resume (BP.Manager) ;                    /* Resume the manager */
12.         Deschedule (BP.list) ;      /* Wait breakpoint end. Resumed by manager */
13.         }
```

III.2.3. ALGORITHME M DU MANAGER ET DU SUPERVISEUR DE POINT D'ARRÊT.

L'algorithme du manager et du superviseur est commun pour permettre à chaque noeud de devenir à son tour superviseur avec le même processus. Cependant nous allons décrire leurs comportements séparément.

L'instruction *Alt* permet de bloquer un processus pour recevoir un message sur une liste de canaux. En cas de présence de plusieurs communications, l'instruction procède à un choix aléatoire. Les messages sont reçus dans un canal bufferisé.

Le superviseur reste en attente d'une demande d'arrêt pouvant venir d'un manager ou d'un processus local. Après réception de la demande, il la diffuse à tous les managers. Puis il active le point d'arrêt et réveille l'éventuel demandeur local. Ensuite, le superviseur attend la requête de

terminaison du point d'arrêt émise par le responsable, et la diffuse aux managers. S'il reste encore des actions à terminer, il attend que la dernière action le réveille (22-23 vs I10-11). Puis le superviseur attend des managers l'acquittement collectif de la fin des actions sur chaque noeud, et annule les demandes d'arrêt pendantes. Le superviseur accorde un délai aux processus qui communiquaient avec les adhérants pour être remis dans la file d'ordonnancement avant la fin du point d'arrêt. Enfin, il diffuse la confirmation de l'arrêt, réveille les processus adhérants, et détermine le prochain noeud superviseur.

Le manager reste en attente d'une demande d'arrêt venant du superviseur ou d'un processus local. Si celle-ci vient d'un processus local, il la transmet au superviseur sans se bloquer, et il attend le message diffusé par ce dernier. Le manager active le point et éventuellement envoie un signal d'acquittement au demandeur local. Ensuite, le manager attend la demande de terminaison du point d'arrêt émanant du superviseur, et annule l'arrêt. S'il reste encore des actions à terminer il attend que la dernière action le réveille (22-23 vs I10-11). Ensuite, le manager envoie au superviseur un acquittement sans se bloquer, et il attend de celui-ci un message confirmant la fin de l'action distribuée. Enfin, le manager réveille les processus adhérants, et détermine le prochain noeud superviseur.

```
1.  Process manager () {
2.      MESSAGE msg ; int tmp ;
3.      int my_node = get_node_number () ;
4.      FOREVER {
5.          Alt (BP.Local_Req, BP.Global_Req) ;          /* Receive any of local or global request
    */
6.          if (my_node == BP.Supervisor)                /* Supervisor behavior */
7.              Broadcast (BP, HALT);                    /* Broadcast the request*/
8.          else if (Local_Req) {                        /* Manager bihaviour */
9.              asend (BP.Global_Req) ;          /* not blocking: forward request to supervisor*/
10.             recv (BP.Global_Req);            /* Receive request broadcast by supervisor */
11.             }
12.         set_breakpoint (BP) ;
13.         if (BP.Local_Req)
14.             Signal (BP_ SIGNAL);                     /*Resume the local initiator*/
15.         if (my_node == BP. Supervisor) {
16.             recv (BP.Resume);                /* Receive resume request from initiator*/
17.             Broadcast (BP.Resume) ;          /* Broadcast resumption req to managers */
18.             }
19.         else
20.             recv (BP.Resume) ;               /*Receive broadcast resumption msg */
21.         BP.status = END ;
22.         if (BP.ctr)          /* Distributed action is not yet terminated on the node */
23.             Deschedule (BP.Manager) ;        /* Wait for the last user process */
24.         if (my_node == BP. supervisor) {     /* Gather completions and broadcast ack. */
25.             Gather (BP.Gather) :                     /* Issued by all managers */
26.             Cancel (BP.Global_Req) ;  /* Remove pending messages from bufferized channel */
27.             sleep (LATENCE_TIME) :           /*Inserted in the scheduler queue at resumpt°*/
28.             Broadcast (BP.End) :                     /* Acknowledge the node managers */
29.             }
30.         else {                                /* Send completion to supervisor and receive ack. */
31.             asend (BP.Gather) ;              /* Not blocking: send to the supervisor */
32.             recv (BP.End) ;                  /* Receive broadcast ack. from supervisor */
33.             }
34.         BP. Supervisor = compute_next_ supervisor _node () ;   /* Round robin can be used*/
35.         Resume_halted_processes (BP.list) ;
36.         BP.status = FREE ;
```

```
37.     }}
```

III.2.4. ALGORITHME T DE TERMINAISON DU POINT D'ARRÊT.

Cette fonction permet au processus responsable du point d'arrêt d'en demander la terminaison directement au superviseur. Toute tentative faite par un autre processus échoue à cause de l'authentification de la signature.

```
1. Disable_Breakpoint () {
2.          if ((BP.status == ACTIVE) && (BP.Signature == my_signature))
3.                  send_to_supervisor (BP.Resume) ;
4.          }
```

III.3. PREUVE DE CONSISTANCE.

Dans cette section, nous allons démontrer les trois théorèmes suivants.

1. Un seul processus est responsable du point d'arrêt.
2. Deux points d'arrêt ne peuvent pas s'imbriquer ni se recouvrir.
3. L'arrêt est consistent.

Dans la suite nous ferons références aux algorithmes en indiquant leur lettre éventuellement suivie d'un numéro d'instruction dans l'algorithme. Par exemple M12 indique l'instruction n° 12 de l'algorithme M. Rappelons que les fonctions G, I, M et T sont exécutées en mode privilégié, et qu'un processus ne peut y être suspendu qu'aux points de communication.

III.3.1. UNICITÉ DU RESPONSABLE DU POINT D'ARRÊT.

Nous devons démontrer que le point d'arrêt n'est activé que par un seul processus dans tout le système.

Pour un noeud. Le mode d'exécution de la fonction G servant à demander l'activation d'un point d'arrêt permet au premier processus p qui trouve le status à FREE de positionner ce dernier à BUSY (G3-4). Ce faisant, tous les autres processus du noeud voulant exécuter G vont échouer sur G3 à cause du status modifié par p. Tant que le status n'est pas revenu à FREE, aucun processus ne peut franchir la séquence G3-4. Or le status n'est remis à FREE que par le manager quand le point d'arrêt est terminé. Par conséquent, aucun processus ne peut demander l'activation d'un point d'arrêt sur le noeud dès que p a franchi G3-4. Le processus p est donc le seul demandeur possible, et donc le seul responsable potentiel sur le noeud.

Pour tous les noeuds. Un point d'arrêt ne peut être activé (status égal à ACTIVE) que sur ordre du superviseur qui a reçu une demande d'activation venant d'un processus local ou d'un manager (M5). Une seule demande est prise en compte et diffusée (M7) par le superviseur. Le demandeur ainsi choisi deviendra l'unique responsable du point d'arrêt. Or les managers n'activent le point d'arrêt (M12) que lorsqu'ils reçoivent (M10) le message diffusé par le superviseur. Donc la même demande d'arrêt est reçue et prise en compte par tous les managers. Noter que quand un manager reçoit le message diffusé, il réveille le processus qui aurait fait une demande de point d'arrêt (M13-14). Mais sur tout le système, un seul de ces demandeurs aura été élu responsable du point d'arrêt par le superviseur. Son identité correspond à la signature que porte le message diffusé. Seul ce processus sortira de G avec la valeur 1 (G7-8).

Conclusion. Chaque noeud ne peut avoir qu'un seul demandeur. Parmi tous les demandeurs des N noeuds, un seul deviendra le responsable du point d'arrêt : celui dont le message aura été diffusé. CQFD.

III.3.2. ABSENCE D'IMBRICATION ET DE RECOUVREMENT DE POINTS D'ARRÊT.

Deux points d'arrêt ne peuvent s'imbriquer ou se recouvrir que si un point d'arrêt démarre pendant qu'un autre est actif. Pour montrer que cela n'est pas possible avec notre système, il suffit de démonter qu'un point d'arrêt ne peut pas démarrer tant qu'un autre est actif. Nous avons vu qu'une seule demande est à l'origine du point d'arrêt, et que les autres demandes sont rejetées. Analysons donc la fin d'un point d'arrêt.

Pour déclencher la terminaison d'un point d'arrêt, le responsable du point d'arrêt envoie un message au superviseur T3. A la réception de ce massage M15-16, le superviseur le diffuse vers les managers M17. Chaque manager à la réception met le status à END. Ensuite, les managers et le superviseur attendent la fin de l'action distribuée locale M22-23, chacun étant réveillé par la dernière action à se terminer I10-11. Puis les managers acquittent la réception de ce message M31, signifiant la fin de l'action distribuée. Le superviseur attend la réception de tous les acquittements M25, c'est à dire la fin de l'action distribuée avant de diffuser M28 la terminaison effective du point d'arrêt. Chaque manager recevant ce dernier message réveille alors les processus qui étaient arrêtés (M35 vs I12) et libère le status M36. Noter que le superviseur fait la même chose après la diffusion. On remarque donc que le superviseur n'autorise M28 le réveil des adhérants que quand tous les noeuds ont terminés leurs actions. Le status n'est effectivement remis à FREE que quand tous les managers reçoivent cette autorisation, indiquant à chacun que tout le monde a terminé. Alors seulement un nouveau processus p peut franchir G3-4.

En conclusion, aucun processus p ne peut franchir G3-4 tant que tous les managers n'ont pas acquitté la terminaison de l'action distribuée, c'est à dire du point d'arrêt. Donc aucun point d'arrêt ne peut démarrer tant que le point d'arrêt en cours ne s'est pas terminé. CQFD.

III.3.3. ARRÊT CONSISTANT.

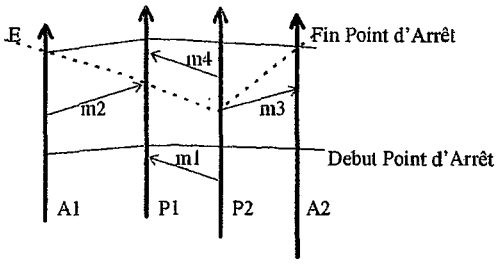
Nous allons montrer que l'arrêt est consistant à la fois du point de vue de l'état global et des événements collectés.

Consistance de l'état global.

Chaque adhérant exécute l'algorithme I qui prend en paramètre le pointeur d'une fonction f à exécuter. La fonction f est propre à chaque processus et est destinée à collecter l'état de ce processus au moment où il rentre dans le point d'arrêt. En fait, c'est le processus lui même qui collecte son propre état à travers cette fonction. C'est donc une mise en oeuvre de la proposition de Chandy-Lamport. CQFD.

Consistance de l'ensemble des événements.

Soit E1 l'ensemble constitué des événements qui se sont produits avant le point d'arrêt. Soit E2 le sous ensemble des événements qui se sont produits pendant le point d'arrêt et qui sont liés par la relation de précédance causale [Ada] [Bou] [Mat] à un événement d'un adhérant. Soit $E = E1 \cup E2$. La figure ci-dessous montre comment E peut être obtenu pour 2 adhérants A1 et A2 et deux non adhérants P1 et P2. Les pointillées indiquent les limites de la coupe E.



Théorème. L'ensemble E constitue une coupe consistante.

Preuve.

D'après un corollaire présenté dans [Ada], pour établir cette preuve, il nous suffit de démontrer qu'à toute réception d'un message prise en compte dans E, correspond une émission du même message elle aussi prise en compte dans E. Rappelons qu'une communication est considérée comme terminée quand l'un au moins des processus impliqués est remis dans la file d'ordonnancement.

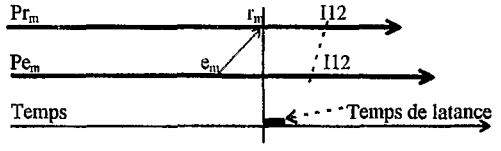
Si e est un événement, alors Pe désigne le processus qui a produit l'événement e; et Me désigne le manager (ou le superviseur) du noeud sur lequel l'événement e s'est produit. Si m est un message, alors e_m désigne l'événement correspondant à l'émission de m, et r_m l'événement correspondant à la réception. Pe_m et Pr_m désignent respectivement le processus émetteur et récepteur de m.

Soit $\mathcal{M}$ l'ensemble des messages échangés lors de l'exécution, nous devons montrer que E est une coupe consistante si:

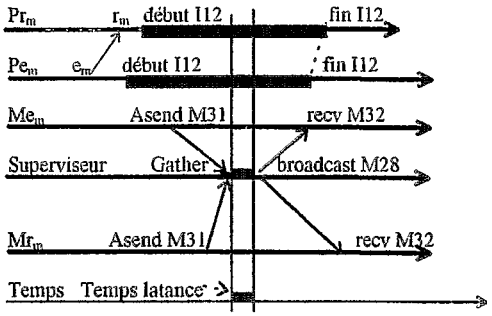
$$\forall\, m \in \mathcal{M},\ r_m \in E \Rightarrow e_m \in E.$$

Nous avons trois cas de figures. Soit les processus Pe_m et Pr_m sont tous les deux adhérants, soit l'un ou bien l'autre est adhérant.

1. Supposons que Pe_m et Pr_m ont tous les deux adhérés au point d'arrêt. Puisque $r_m \in E$, alors r_m s'est produit avant la suspension de Pr_m en I12 (fig. ci-dessous). L'événement e_m s'est terminé Tl (temps de latence) unités de temps au plus tard après r_m. Or le superviseur impose une attente au moins égale au temps de latence (M27) à tous les adhérants avant d'autoriser leur réveil (M28, M35). Comme Pe_m est aussi adhérant, il passe un temps supérieur au temps de latence dans I12. Donc e_m s'est aussi produit avant la suspension de Pe_m en I12. Il s'en suit que: $e_m \in E$.



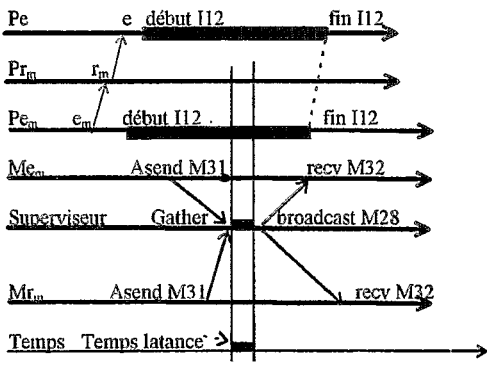
2. Supposons maintenant que Pr_m seulement ait adhéré au point d'arrêt (fig. ci-dessous). Comme nous avons $r_m \in E$, le message m a été reçu avant que Pr_m n'exécute I12. Nous devons démontrer que m a été émis avant la fin du point d'arrêt; c'est à dire que : $e_m \in E$. Cela n'est possible que si Pe_m a été remis dans la file d'ordonnancement et s'est exécuté au moins une fois avant que le point d'arrêt ne se termine afin de prendre en compte e_m.



Comme $r_m \in E$, et Pr_m est adhérant, alors r_m se produit avant que Pr_m ait exécuté I12. Ce qui signifie que le manager Mr_m a activé le point d'arrêt. Quand ces événements se produisent, nous ne pouvons rien conclure de Me_m. Cependant Me_m et Mr_m sont synchronisés par le superviseur (M31-32). A cause de cette barrière de synchronisation imposée par le superviseur, nous savons que Pr_m ne termine I12 que si le manager Me_m a exécuté M31. C'est à dire que le récepteur Pr_m après avoir terminé la réception de m, ne peut continuer son exécution en I13 que si le manager Me_m du noeud émetteur de m a acquitté le superviseur.

Or après avoir reçu tous les acquittements, le superviseur attend au minimum le temps de latence (M27). Temps pendant lequel le mécanisme d'ordonnancement a remis Pe_m dans la file d'ordonnancement puisque la communication est terminée. Après ce temps d'attente, le manager Me_m est remis lui aussi en fin de file d'attente. Ce qui signifie que l'événement e_m se termine avant la fin du point d'arrêt. Donc: $e_m \rightarrow M35$; et comme: $M35 \in E$, nous avons $e_m \in E$.

3. Supposons que Pe_m seulement ait adhéré au point d'arrêt. Comme $r_m \in E$, nous devons démonter que $e_m \in E$. Du fait que $r_m \in E$, nous savons par définition de E, qu'il existe un événement e appartenant à un adhérant Pe tel que : $r_m \rightarrow e$, $e \in E$. L'événement e s'est produit avant la terminaison du point d'arrêt, c'est à dire que e s'est produit avant que Pe n'exécute I12. Nous savons que Me est synchronisé avec Me_m. A la fin de l'action distribuée, le superviseur impose à chaque manager une attente égale au temps de latence (M27) avant de les autoriser de réveiller les adhérants (M35).



Cela signifie qu'après e, il s'est écoulé au minimum le temps de latence, avant que le réveil synchronisé ne soit autorisé. Or nous savons que r_m est antérieur à e ($r_m \rightarrow e$). Donc à la fin de r_m, il s'est écoulé au minimum le temps de latence avant que le réveil ne soit autorisé. Ce qui implique que (1) : e_m s'est

terminé avant que le réveil ne soit autorisé. Comme Pe_m est un adhérant (2) : il attend le réveil dans I12. Les propositions (1) et (2) nous permettent de conclure que e_m est antérieur à l'instruction I12 exécutée par Pe_m pour l'attente. Donc: $e_m \rightarrow I12 \Rightarrow e_m \in E$ (où I12 est exécuté par Pe_m).

Nous avons donc démontré que: $\forall m \in \mathcal{M}, r_m \in E \Rightarrow e_m \in E$.

Nous pouvons donc conclure que E est une coupe consistante. E étant une coupe consistante, donc l'arrêt ayant permis d'obtenir E est consistant du point de vue des événements; la proposition 2 est ainsi vérifiée. Comme les propositions 1 et 2 sont toutes les deux vérifiées, nous pouvons conclure que l'arrêt global est consistent. CQFD.

III.4. EVALUATION.

Nous allons supposer dans cette évaluation qu'une diffusion engendre N-1 communications, donc N-1 messages; où N est le nombre de noeuds. De même, une communication entre un noeud et le superviseur peut au pire traverser les N-1 noeuds (cas de routage).

- Chaque noeud peut abriter un demandeur de point d'arrêt. Donc un message pourra être échangé sur chaque noeud entre le demandeur (G5) et le manager (M5); coût: N messages. Le manager ayant reçu une demande va envoyer un message au superviseur; soit N-1 communications pour convoyer ce message; coût $(N-1)^2$. Nous aurons donc au pire $N + (N-1)^2$ messages pour demander un point d'arrêt.
- Le superviseur diffuse un message qui coûtera N-1. Puis sur chacun des N noeud un demandeur potentiel est réveillé, soit N messages. Coût total: **2N - 1**.
- Quand le responsable décide de terminer l'arrêt, il envoie un message au superviseur dont la transmission peut engendrer **N-1** communications.
- Le superviseur diffuse un message de terminaison de l'arrêt; coût total: **N - 1**.
- Les N-1 managers envoient chacun un acquittement au superviseur; coût: $(N-1)^2$. Enfin, le superviseur diffuse un message final. Coût total: **N(N-1).**

Nombre maximal de messages échangés: $2N^2 + 2N - 2$. Ce coût est indépendant du nombre de processus et est lié à la topologie d'interconnexion des noeuds.

V. CONCLUSION.

Le système que nous venons de décrire réalise un arrêt global sélectif. Il permet en particulier d'arrêter des groupes de processus individuellement et par les dépendances de communication. Le nombre de messages échangés lors d'un point d'arrêt est borné par une constante fonction du nombre de processeurs. Ce nombre Ce coût est indépendant du nombre de processus dans la machine. Le type de réseau d'interconnexion des processeur peut améliorer cette constante. Par exemple dans un hypercube de degré $\log_2 N$, nous aurons au pire $5N - 3 + (2N - 1)\log_2 N$ messages.

L'algorithme est cependant sensible aux pannes. Il repose donc sur la fiabilité de la machine parallèle. Cependant en modifiant l'instruction *gather* de collecte des acquittements (M63) en y introduisant par exemple des temporisations, il est possible de détecter de longs temps d'attente trahissant une panne.

L'hypothèse que nous avons prise sur le temps de latence est réaliste du fait qu'elle est vraie dans les machines à base de transputers. Dans d'autres, la transmission de message est assurée en s'appuyant sur la gestion des interruptions. Il suffit d'attribuer le niveau de priorité le plus élevé à la gestion des

communications pour que l'hypothèse du temps de latence soit valide. Le présent algorithme sera prochainement implémenté sur un Tnode TN310 de chez Telmat Informatique doté de 32 processeurs T9000 actuellement disponible à l'IAI.

Références.

[Ada] Adam M., Hurfin M., Raynal M., Plouzeau N.
Distributed Debugging Techniques
RR. N° 1459 INRIA, Rennes

[Bem] Bemmerl T., Wismüller R.
On-Line Distributed Debugging on Scalable Multiprocessor Architecture
Future Generation Computers systems Aug. 1995 Vol. 11 N° 4-5 PP 363-374

[Bou] Bougé Luc
Repeated Snapshot in Distributed Systems with Synchronous Communications and their Implementation in CSP
Theorical Computer Science N° 49, 1987, PP. 145-69

[Bur] Buhr P.A., Fortier M., Coffin M.H
Monitor Classification
ACM Press Computing Surveys PP 63-108, 1995

[Cha] Chandy K.M., Lamport L.
Distributed Snapshots: Determining Global States of Distributed Systems
Trans. on Computing Systems 3, 1, Feb. 1985, PP. 63-75

[Fei] Feitelson D., Rudolpf L.
Parallel Job Scheduling: Issues and Approaches
Job Scheduling Strategies for Parallel Processing, IPPS'95 Workshop, Santa Barbara, CA USA Apr. 25, 1995 PP 1-18

[Hab] Haban D. Weigel W.
Global Events and Global Breakpoints in Distributed Systems
21th Hawai International Conference on System Science. Jan. 1988, PP. 166-175

[INM] INMOS
The T9000 Transputer, Product Overview Manual
INMOS, First Edition 1991

[Jon] Jone S.H. et al.
BugNet : a Real Time Distributed Debugging Systems
Proc. 6th Int. Symp. on Reliability in Distributed Soft. and DB Systems
Williamsburg, Va, PP. 56-65, March 1987

[Leb] Leblanc T.J., Mellor-Crummy J.M.
Debugging Parallel Programs with Instant Replay
Transaction on Computers, Vol. C 36, N° 4, PP. 471-482, April 1987

[Mat] Mattern F.
Virtual Time and Global States of Distributed Systems
Departement of Computer Science, Univ. of Kaiserslautern, D 6750 Kaiserslautern, Germany

[Mil] Miller B.P., Choi J.D.
Breakpoints and Halting in Distributed Programs
Proc. 8th Int'l Conf. on Distributed Computing Systems, CS Press, Los Alamitos Calif. 1988, PP. 316-323

[Mou] Moukèli Pierre
Un Système de Construction de Points d'Arrêt Globaux pour les Programmes Parallèles
Proc. of the 1st African Conference on Research in Computer Science CARI'92, Yaoundé, Cameroon, 1992, Oct. 14-20, PP. 705-716.

[Nai] Naimi Mohamed
Global Stability Detection in the Asynchronous Distributed Computations
IEEE, Proc. Work. on the Futrure Trends of Distributed Computing Systems in 1990, Hong Kong, 14-16 Sept. 1988, PP. 87-92

[Sch] Schwiegelshohn U.T., John Wolf Joel
Preemptive Scheduling of Parallel Tasks
RC 20104 IBM Research Centers, Yorktown, San Jose, 06 1995, 1995

[Str] Strong D. M., Miller S. M.
Exceptions and Exception Handling in Computerized Information Processes
ACM Transactions on Information Systems Apr. 1995 Vol. 13 N° 2 PP 206