

Algorithmique répartie

Problèmes génériques

Gérard Padiou

Département Informatique et Mathématiques appliquées
ENSEEIH

Janvier 2008



plan

- 1 Exclusion mutuelle
 - Le problème
 - Classes d'algorithmes
 - Un algorithme(Ricart-Agrawala)
- 2 Interblocage
 - Le problème
 - Caractérisation de l'interblocage
 - Un algorithme de détection(Chandi,Misra,Haas)
- 3 Problème de la terminaison
 - Spécification du problème
 - Notion de calcul diffusant
 - Algorithme des compteurs (Mattern)
 - Algorithme des crédits (Mattern)



Plan

- 1 Exclusion mutuelle
 - Le problème
 - Classes d'algorithmes
 - Un algorithme(Ricart-Agrawala)
- 2 Interblocage
 - Le problème
 - Caractérisation de l'interblocage
 - Un algorithme de détection(Chandi,Misra,Haas)
- 3 Problème de la terminaison
 - Spécification du problème
 - Notion de calcul diffusant
 - Algorithme des compteurs (Mattern)
 - Algorithme des crédits (Mattern)



Spécification du problème

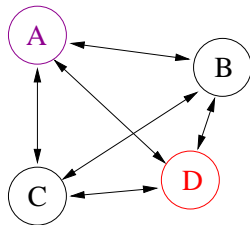
```
process P(int i) :  
    ... ; Entrer(i); <SC> ; Sortir(i); ...
```

Diagramme d'état



- Un **seul** processus au plus en exclusion ;
- Sûreté : pas d'interblocage ;
- Vivacité : Tout processus finit par entrer ;
- Protocole : Tout processus finit par sortir ;

Réseau maillé fiable



D est en exclusion.
A est candidat

Principes de base

Règle de base

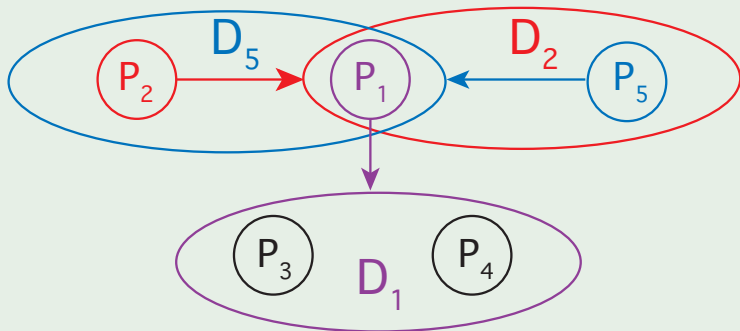
Tout processus candidat doit demander à d'autres processus la **permission** d'entrer en exclusion

- À tout P_i on associe un ensemble D_i contenant les processus à contacter ;
- **Condition nécessaire** : $\forall i \neq j : j \in D_i \vee i \in D_j$
- Deux types de permissions :
 - **individuelles** : un processus donne son autorisation selon son propre état ;
 - **d'arbitre** : les processus s'échangent des permissions préexistantes en nombre fixé.
- **Objectif** : Minimiser les ensembles D_i .

Permissions individuelles

Exemple

Compléter : D_3, D_4 ?



Permissions individuelles : au pire ...

Ricart et Agrawala

Hypothèses

- Chaque processus sait qu'il existe N processus ;
- Réseau de communication fiable et maillé ;
- Pas de défaillance de processus.

Solution

- Utilisation de permissions individuelles avec :

$$\forall i : D_i = \{Tous\} - \{i\}$$

- Ordonnancement des requêtes par datation.

Algorithme de Ricart et Agrawala

Principes

- Requêtes d'entrée totalement ordonnées :
 - ☞ Utilisation d'horloges de Lamport
- Chaque processus P_i candidat ou en exclusion connaît la date de sa requête courante $Date(R_i)$;
- Un candidat entre en exclusion s'il a obtenu les permissions de tous les autres ;
 - ☞ il possède alors la requête la plus ancienne
- Revient à vérifier que la requête R_i d'un processus P_i est la plus vieille requête des processus candidats ou en exclusion :

$$\forall k : P_k.Etat \neq hors \Rightarrow Date(R_i) \leq Date(R_k)$$

Un algorithme de base

Algorithme de Ricart-Agrawala

```
process P(i : 0..N-1) {
  type Etat = {hors,candidat,exclusion}; Etat EC = hors ;
  Date hloc=new Date(i,1) ; Date drLoc ;
  Set<int> Att=new EnumSet<int>() ;Set<int> D=EnumSet.range(0,N-1).remove(i) ;
  while (true) {
    select {
      when (EC==hors)⇒ // hors → candidat
        EC=candidat ; drLoc=hloc.Top() ;
        for(int p : D) send Rq(i,drLoc) to p ;
      || when (EC==exclusion)⇒ // exclusion → hors
        for(int p : Att) send Perm(i) to Pp ; Att.clear() ;EC=hors ;
      || when (EC==candidat) ⇒ receive Perm(p) // candidat ? → exclusion
        D.remove(p) ; if(D.empty()) EC=exclusion ;
      || receive Rq(p,dr) ;
        hloc.Recaler(dr) ;
        if(EC !=hors && Date.pred(drloc,dr))Att.add(p) ;else send Perm(i) to Pp ;
    } // select
  } // while
}
```

Permissions d'arbitres

Definition

Condition nécessaire : \exists arbitre commun : $\forall i \neq j : D_i \cap D_j \neq \emptyset$

Exemple

<i>i prend pour arbitre :</i>	1	2	3	4	5
1		•	•		
2			•	•	
3		•		•	
4		•		•	
5		•	•		

Permissions d'arbitres

Exemple

Optimisation :

- partir d'une matrice (ici 9 sites) :

1	2	3
4	5	6
7	8	9

$$D_5 = \{2, 4, 6, 8\}$$

- Tout processus utilise les arbitres de sa colonne et de sa ligne
- Tout processus utilise 4 arbitres : ici $|D_i| = 4$
- Tout arbitre appartient au même nombre d'ensembles D_i : ici 4

Plan

- 1 Exclusion mutuelle
 - Le problème
 - Classes d'algorithmes
 - Un algorithme(Ricart-Agrawala)
- 2 **Interblocage**
 - Le problème
 - Caractérisation de l'interblocage
 - Un algorithme de détection(Chandi,Misra,Haas)
- 3 Problème de la terminaison
 - Spécification du problème
 - Notion de calcul diffusant
 - Algorithme des compteurs (Mattern)
 - Algorithme des crédits (Mattern)

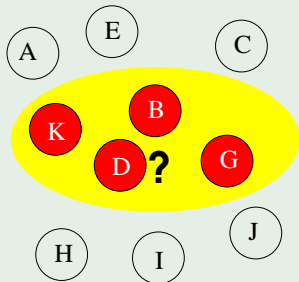


Spécification du problème

Détection d'une propriété stable

- Interblocage dû aux communications : attente de la réception d'un message
- Un processus est-il définitivement bloqué ?
- Sûreté : pas de fausse détection ;
- Vivacité : Un processus bloqué finit par le savoir ;

Exemple



D s'interroge ?

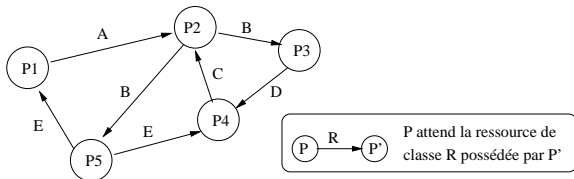
K,B,D,G en interblocage

Caractérisation de l'état d'interblocage

Notion de composante fortement connexe terminale (CFCT)

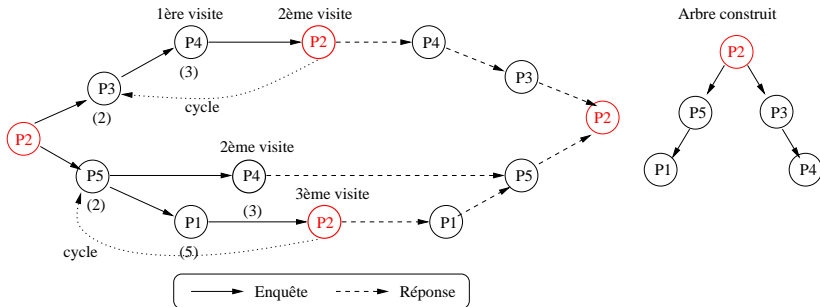
Un sous-graphe G' d'un graphe $G = \{S, A\}$ est une CFCT ssi il existe un chemin entre tout couple de sommets de G' et si tout sommet de G' a ses successeurs dans G' :

$$\forall s, s' \in G' : \exists s \xrightarrow{*} s' \wedge \forall s \in G' : \text{succ}(s) \neq \emptyset \wedge \text{succ}(s) \subset G'$$



Algorithme : calcul diffusant et arbre de contrôle

de K.Mani Chandy, J. Misra, Laura M.Haas



Propriétés

- Terminaison de la construction de l'arbre $\Rightarrow P_2$ est interbloqué
- Pas de terminaison : Il faudra recommencer...

Plan

- 1 Exclusion mutuelle
 - Le problème
 - Classes d'algorithmes
 - Un algorithme(Ricart-Agrawala)
- 2 Interblocage
 - Le problème
 - Caractérisation de l'interblocage
 - Un algorithme de détection(Chandi,Misra,Haas)
- 3 **Problème de la terminaison**
 - Spécification du problème
 - Notion de calcul diffusant
 - Algorithme des compteurs (Mattern)
 - Algorithme des crédits (Mattern)

Détection d'un propriété stable

Spécification

- Propriété **stable** à détecter :

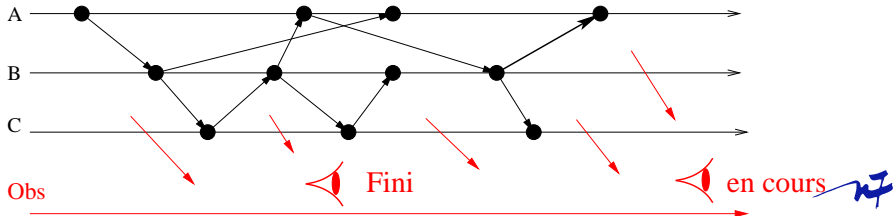
Tous les processus sont passifs **ET** pas de message en transit.

- Sûreté : Pas de **fausse détection** :

$$Term \Rightarrow (\forall i :: P_i.passif \wedge EnTransit = \emptyset)$$

- Vivacité : La terminaison **fini** par être détectée :

$$(\forall i :: P_i.passif \wedge EnTransit = \emptyset) \rightsquigarrow Term$$

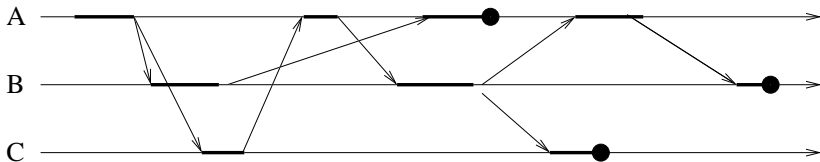


Definition

Calcul diffusant :

- Un processus au moins émet un ou plusieurs messages ;
- Puis, tous les processus adoptent le même comportement :

```
loop { /* un pas de calcul */  
      recevoir( $m$ ) ;  
      traiter  $m$  ;  
      envoyer 0 à  $N - 1$  messages ;  
}
```

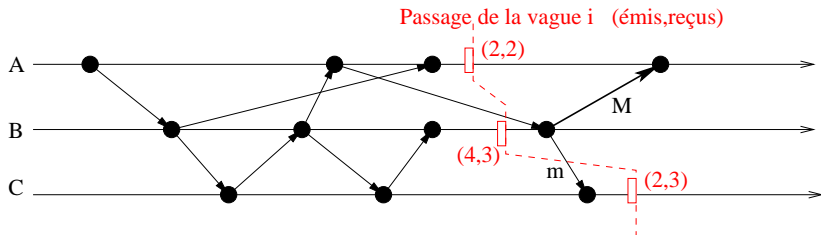


Handwritten signature

Algorithme des compteurs (Mattern)

Principe

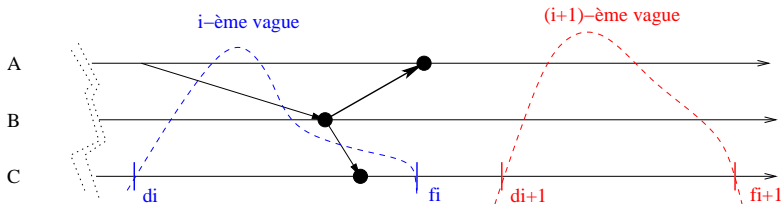
- Terminaison $\equiv E(t) == R(t)$ **MAIS** impossible à évaluer ;
- Approche : Compteurs **locaux** des messages émis et reçus ;
- Mécanisme de **vague** pour collecter les valeurs des compteurs ;
- La vague i collecte $R_i = \sum r_i$ et $E_i = \sum e_i$.



Algorithme des compteurs (Mattern)

Détection de la terminaison

- nécessite **deux** vagues successives ;
- Terminaison si : $R_i == E_{i+1}$ ($\Rightarrow \exists t < d_{i+1} : E(t) == R(t)$)
- Détection avec un retard d'au + la durée de la dernière vague ;



Algorithme des crédits (Mattern)

Principe

- Le processus initial possède un crédit de 1 ;
- Le crédit courant est **partagé** entre les messages émis ;
- Un processus **rend** son crédit s'il n'envoie pas de message ;
- Terminaison lorsque la **somme** collectée égale 1.

