Systèmes et algorithmes répartis

Problèmes génériques

Philippe Quéinnec, Gérard Padiou

ENSEEIHT
Département Sciences du Numérique

13 octobre 2020

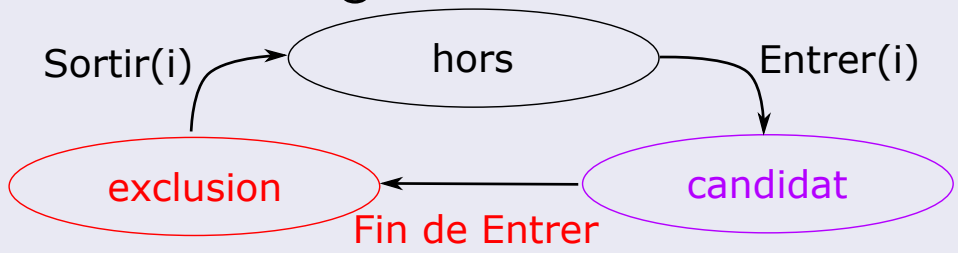
Plan

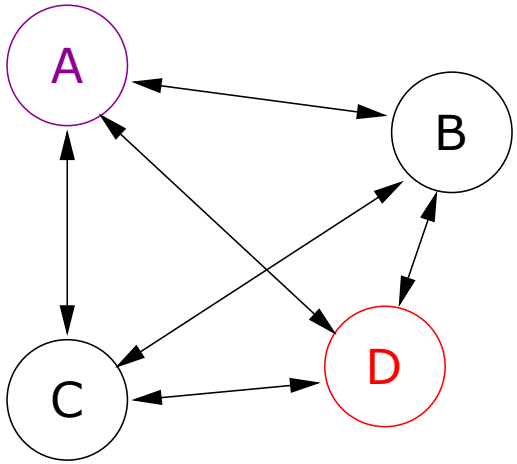
Plan

Spécification du problème

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

Diagramme d'état





D est en exclusion
A est candidat

- Sûreté : Un processus au plus en exclusion
  $\forall i, j :: P_i.exclusion \wedge P_j.exclusion \Rightarrow i = j$
- Vivacité faible : pas d'interblocage (certains candidats finissent par entrer)
- Vivacité forte : Tout candidat finit par entrer
- Protocole : Tout processus en exclusion finit par sortir

[Précis 4.1 pp.63–65]

Élection vs exclusion mutuelle

Problèmes similaires. . .

Isoler un processus parmi tous : introduire une asymétrie

. . . mais bien différents

- Élection d'un quelconque des processus mais exclusion mutuelle parmi les candidats
- L'élection est définitive mais l'exclusion mutuelle se termine et se transmet $\Rightarrow$ évolution dynamique

Algorithme à base de jeton circulant

Algorithme basé sur le contrôle d'un objet circulant

- Anneau logique (indépendant de la structure du réseau physique) : chaque site a un successeur
- Jeton circulant :
  - un site non demandeur transmet le jeton à son successeur
  - un site demandeur attend le jeton pour obtenir l'exclusion mutuelle
  - un site qui sort d'exclusion mutuelle transmet le jeton à successeur

Propriétés

- Sûreté : unicité du jeton
- Vivacité : intégrité de l'anneau (existence et circulation du jeton)

Algorithme à base de jeton circulant

```
Process P(i : 0..N-1) {
 type Etat = {hors,candidat,exclusion}
 Etat EC ← hors;
 bool jeton ← (i = 0); //jeton initialement sur site 0
 on (EC = hors) :                          // hors → candidat
     EC ← candidat;
 on (EC = candidat ∧ jeton) : // candidat → exclusion
     EC ← exclusion;
 on (EC = exclusion) :                     // exclusion → hors
     EC ← hors;
     send MsgJeton to P_{i⊕1}
     jeton ← false;
 on reception MsgJeton :                   // réception jeton
     jeton ← true;
 on jeton ∧ EC = hors :                    // transmission jeton
     send MsgJeton to P_{i⊕1}
     jeton ← false;
}
```

Algorithmes à base de permission

Un 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
- Correction : $\forall i \neq j : j \in D_i \vee i \in D_j$
- Deux types de permissions :
  - individuelles : un processus donne son autorisation selon son propre état $\Rightarrow$ n'engage que lui
  - d'arbitre : les processus s'échangent des permissions préexistantes en nombre fixé $\Rightarrow$ engage tous les processus qui dépendent de lui
- Objectif : Minimiser les ensembles D_i

1. *An Optimal Algorithm for Mutual Exclusion in Computer Networks*, Glenn Ricart and Ashok K. Agrawala. Communications of the ACM, January 1981.

Permissions individuelles : Ricart et Agrawala ♪♪♪

Hypothèses

- Chaque processus connaît l'identité des N processus
- Réseau de communication fiable et maillé (non FIFO)
- Pas de défaillance de processus

Solution

- Utilisation de permissions individuelles avec :
  $\forall i : D_i = Tous - \{i\}$
- Ordonnancement des requêtes par datation

Messages

- Message de requête : le site i demande l'autorisation à j
- Message d'autorisation : le site j donne son autorisation à i
- Pas de message de refus : le refus est temporaire

Algorithme de Ricart et Agrawala

Principes

- Requêtes d'entrée totalement ordonnées :
  $\Rightarrow$ 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
  $\Rightarrow$ 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 :
  $\forall k : P_k.Etat \neq hors \Rightarrow Date(R_i) \leq Date(R_k)$

Algorithme de Ricart-Agrawala

```
Process P(i : 0..N-1) {
 type Etat = {hors,candidat,exclusion}; Etat EC ← hors;
 Date hloc ← new Date(0,i); // horloge locale
 Date dr; // date de la requête de ce site
 Set<int> Att ← ∅;  // sites lui ayant demandé l'autorisation
 Set<int> D;        // sites dont i attend l'autorisation
 on (EC = hors) : // hors → candidat
       EC ← candidat;
       D ← 0..N-1 \ {i};
       dr ← hloc.Top();
       for k ∈ D : send Request(i,dr) to P_k;
 on reception Request(p, d) :
       hloc.Recaler(d);
       if (EC ≠ hors ∧ dr < d) then Att ← Att ∪ {p};
       else send Perm(i) to P_p;
 on (EC = candidat) ∧ reception Perm(p) : // candidat → excl
       D ← D \ {p};
       if (D = ∅) then EC ← exclusion;
 on (EC = exclusion) : // exclusion → hors
       for k ∈ Att : send Perm(i) to P_k;
       Att ← ∅; EC ← hors;
```

Permissions d'arbitres

Principe

Obtenir la permission de tous les arbitres contactés.

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

Exemple

i prend pour arbitre :	1	2	3	4	5
1		•	•		
2			•	•	
3		•		•	
4		•		•	
5		•	•		

Note : 1 et 5 ne sont pas arbitres

Quorums

Définition

Un système de quorum est un ensemble d'ensembles, tel que tout couple d'ensembles a une intersection non vide :
$\mathcal{Q} = \{Q_1, \ldots, Q_n\}, \forall i, j : Q_i \cap Q_j \neq \emptyset$
Un quorum Q_i est responsable pour l'ensemble total.

Idéalement :

- Effort identique : Tous les quorums ont la même taille : $\forall i : |Q_i| = K$
- Responsabilité identique : Tous les sites appartiennent au même nombre de quorums : $\forall i : |\{j : i \in Q_j\}| = D$
- Minimalité : $K = D =$ le plus petit possible (théorie : $\lceil \sqrt{n} \rceil$)

Permissions d'arbitres

Construction facile d'un système de quorum quasi optimal :

- Construire une matrice arbitraire, éventuellement en dupliquant certains sites, p.e. pour 14 sites :

1	2	3	4
5	6	7	8
9	10	11	12
13	14	1	2

- Tout processus utilise les arbitres de sa colonne et de sa ligne

$$D_6 = \{2, 5, 7, 8, 10, 14\}$$

- Tout processus utilise au plus $2\lceil\sqrt{n}\rceil - 1$ arbitres
- Tout arbitre appartient à au plus $2\lceil\sqrt{n}\rceil - 1$ ensembles

1. *A $\sqrt{n}$ Algorithm for Mutual Exclusion in Decentralized Systems*, Mamoru Maekawa. ACM Transactions on Computer Systems, May 1985.

Permissions d'arbitres

ATTENTION : pas de miracle...

Modèle parallèle processus $\leftrightarrow$ ressources critiques

- Un processus doit collecter des jetons $\equiv$ ressources critiques
- Problème : risque d'interblocage
  - Solution préventive : ordonner les jetons et les demander en respectant cet ordre
  - Solution dynamique :
    - ordonner les requêtes (approche transactionnelle) et appliquer l'algorithme « *wound-wait* » ou « *wait-die* »
    - nécessite un ordre total sur les requêtes :
      $\Rightarrow$ usage d'horloges de Lamport
    - Risque de livelock