Introduction aux algorithmes répartis

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/people/krakowia

Objectifs et plan

- **Introduction aux algorithmes de base (hors défaillances ; néanmoins on pourra indiquer quelques contraintes induites par la tolérance aux fautes))**
  - Exclusion mutuelle
  - Élection
  - Terminaison
- **Indiquer quelques méthodes générales d'approche**
  - Principe général : introduire des contraintes sur le déroulement des événements qui composent l'algorithmes, pourfaciliter le raisonnement
  - Plusieurs types de contraintes
    - Ordonnancement des événements (divers types d'horloges, déjà vu)
    - Contraintes sur la topologie de l'application (voies d'échange des messages)
      - Anneau virtuel (+ jetons)
      - Arbre
      - Graphe sans circuits

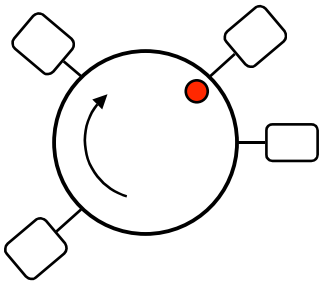
Exclusion mutuelle

- **Rappel des spécifications**
  - Imposer un ordre sur l'exécution des sections critiques
  - Algorithme symétrique, décentralisé
  - Algorithme équitable
  - Garantie de vivacité
- **Solutions**
  - Imposer un ordre global (ex : Ricart-Agrawala et dérivés, déjà vu)
  - Imposer une topologie
    - Anneau virtuel
    - Arbre

Exclusion mutuelle sur un anneau

Inspiration : systèmes physiques de communication (anneau à jeton, ou *token ring*)

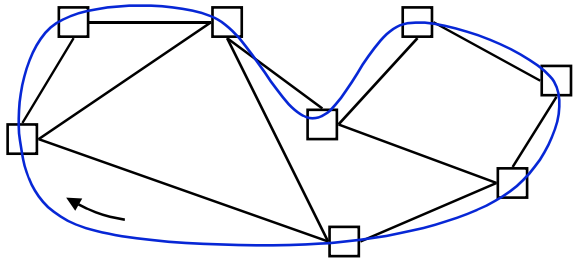
- les stations émettent et reçoivent des messages sur un bus unidirectionnel en anneau
- une station au plus peut émettre à tout instant (brouillage)
- une station ne peut émettre que si elle possède un **jeton** identifié par une configuration unique de bits
- une station garde le jeton pendant un temps limité (vivacité)
- il y a **un jeton et un seul** (invariant à maintenir en cas de défaillance)



Transposition : anneau virtuel

Anneau virtuel

La topologie du système de communication est quelconque. L'anneau virtuel est un réseau superposé au réseau physique (*overlay network*) et utilisant les protocoles du réseau physique



L'anneau virtuel est défini en indiquant pour tout site i son successeur succ[i] ; on peut aussi indiquer son prédecesseur pred[i]

Exclusion mutuelle sur un anneau virtuel

- **Algorithme très simple**
  - **Initialisation**
    - **créer l'anneau virtuel**
    - **créer un jeton (et un seul) sur un site désigné ; lancer la circulation du jeton sur l'anneau**
  - **Entrée en section critique : attendre (jeton)**
  - **Sortie de section critique (site i) : envoyer (jeton, succ[i])**
- **Validité**
  - **La validité de l'algorithme repose sur l'intégrité de l'anneau et sur l'existence et l'unicité du jeton**
    - **sûreté : unicité du jeton**
    - **vivacité : existence du jeton + intégrité de l'anneau (+ durée limitée de la section critique)**
  - **Le bon fonctionnement repose donc sur le maintien de ces propriétés y compris en cas de défaillance**

Maintien de l'intégrité de l'anneau

Hypothèses : réseau fiable ; défaillance des sites ; une défaillance ne partitionne pas le réseau

Principe : auto-surveillance (chaque site surveille son prédécesseur et/ou son successeur (délai de garde : hypothèse de synchronisme)

En cas de détection de défaillance : reconfiguration (si besoin, regénération du jeton)
Après réparation : réinsertion

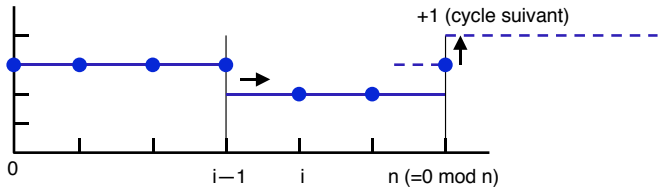
Existence et unicité du jeton

- détection de la perte (délai de garde sur un tour, ou jeton de contrôle)
- perte détectée : choix d'un site (et d'un seul) pour regénérer un jeton : problème de l'élection

Une autre vue du jeton

Un algorithme de vague ("front d'onde") [Dijkstra 74]

Hypothèse : chaque site i possède une variable d'état S[i] (compteur à valeurs entières) et peut consulter le compteur S[i—1] de son prédecesseur
La présence du "jeton" en i est détectée par S[i] ≠ S[i—1] (i ≠ 0)
Le site i exécute la section critique et fait S[i] = S[i—1]
Cas particulier : le site 0. Il incrémente S[0] pour le cycle suivant. Test de présence du jeton : S[i] = S[i—1]



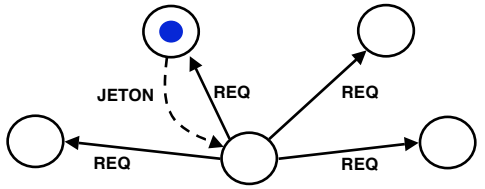
Existence et unicité du "jeton" :
- détection de la perte (délai de garde sur un tour)
- régénération

Algorithmes à jeton

- **De nombreux algorithmes répartis utilisent des jetons**
  - voir plusieurs exemples dans la suite
  - la structure de communication sous-jacente peut être quelconque (anneau, arbre, graphe, non contrainte)
  - le jeton peut être simple (seule est testée sa présence) ou porter une ou plusieurs valeurs
- **Problèmes communs**
  - Garantir la sûreté : unicité (en général) et intégrité du jeton, atomicité des actions
  - Garantir la vivacité : existence du jeton, circulation du jeton, regénération du jeton en cas de perte
  - On cherche à développer des classes d'algorithmes génériques assurant ces propriétés

Exclusion mutuelle : Suzuki-Kasami (1)

Principe : combiner autorisation et jeton (réduire nb de messages)
Un demandeur envoie sa requête à tous les processus; celui qui a le jeton répond

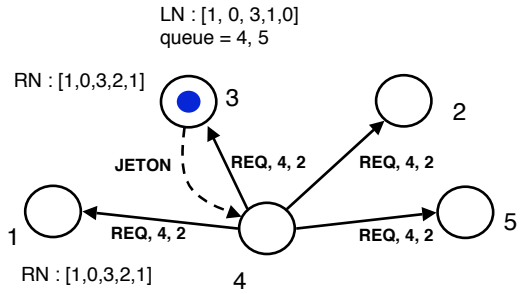


Problème : assurer la vivacité
déterminer si une requête est valide ou périmée (satisfaite)
connaître les requêtes non satisfaites
éviter privation

I. Suzuki, T. Kasami. A Distributed Mutual Exclusion Algorithm, *ACM Trans. on Computer Systems (TOCS)* 3, Nov. 1985

Exclusion mutuelle : Suzuki-Kasami (2)

Structures de données
sur chaque site
RN[i] : numéro d'ordre dernière requête reçue de p_i
sur le jeton
LN[i] : numéro d'ordre dernière exécution de SC par p_i
queue : file d'attente des requêtes non satisfaites pour le jeton



Exclusion mutuelle : Suzuki-Kasami (3)

Demande d'entrée en SC (sur p_i)

```
if  not jeton_présent
      RN[i] = RN[i] + 1
      diffuser(REQ, i, RN[i])
      attendre (jeton_présent)
sc_en_cours = true
entrer en section critique
```

Sortie de SC

```
sc_en_cours = false
LN[i] = RN[i]
for all j I p_j ∉ queue
      if RN[j] == LN[j] + 1
         entrer_queue (j)
if not queue_vide
      k = sortir_queue
      envoyer (k, JETON)
```

Réaction à l'arrivée d'un message sur p_i

• **réception de (REQ, j, n):**

```
RN[j] = max (RN[j], n)
if  jeton_présent and not sc_en_cours
      if RN[j] == LN[j]+1
         envoyer (j, JETON)
         jeton_présent = false
```

• **réception de JETON :**

```
jeton_présent = true
sc_en_cours = true
entrer en section critique
```

Exclusion mutuelle : Suzuki-Kasami (4)

Conclusion

Algorithme simple, combine autorisation et jeton sans imposer de structure particulière au réseau

L'algorithme n'est pas rigoureusement symétrique : le dernier utilisateur du jeton garde le jeton, et il est donc avantagé s'il rentre en section critique avant une nouvelle demande

N messages par entrée en s.c. (ou 0 dans le cas particulier ci-dessus)

La réduction du nombre de messages a comme contrepartie une structure plus complexe pour le jeton (et rend donc plus difficile la regénération du jeton en cas de perte)

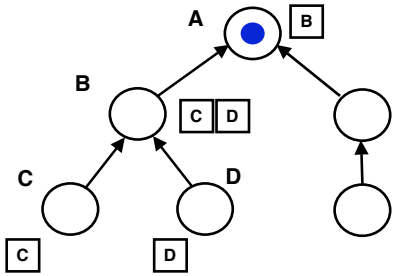
Exclusion mutuelle : Raymond (1)

Principe : organiser les processus en arbre (reconfigurable) ayant pour racine le site qui possède le jeton, orienté vers la racine

Les demandes du jeton (requêtes d'entrée en section critique)
- sont propagées vers la racine
- sont enregistrées dans une queue sur chaque site du trajet

Arbre modifié (inversion du pointeur) à chaque transmission du jeton

Intérêt : nb réduit de messages (~log N)



K. Raymond, A tree-based algorithm for distributed mutual exclusion, *ACM Trans. on Computer Systems (TOCS)* 7(11):61—77, 1989

Exclusion mutuelle répartie (Raymond)

Demande d'entrée en SC (sur p_i)

```
if  not jeton_présent
    if queue_vide
       envoyer(ptr, REQ)
    entrer_queue (i)
    attendre (jeton-présent)
sc_en_cours = true
entrer en section critique
```

Sortie de SC

```
sc_en_cours = false
if not queue_vide
    ptr = sortir_queue
    envoyer (ptr, JETON)
    jeton_présent = false
    if not queue_vide
       envoyer(ptr, REQ)
```

Réaction à l'arrivée d'un message sur p_i (depuis p_j)

réception de REQ:

```
if  jeton_présent
        if sc_en_cours
            entrer_queue (j)
        else
            ptr = j
            envoyer (ptr, JETON)
            jeton_présent = false
else
        if  queue_vide
            envoyer(ptr, REQ)
        entrer_queue (j)
```

réception de JETON :

```
ptr = sortir_queue
if  ptr = i
        jeton_présent = true
else
        envoyer (ptr, JETON)
        if not queue_vide
            envoyer(ptr, REQ)
```

Résumé de l'exclusion mutuelle

L'exclusion mutuelle est un problème simple en centralisé, assez complexe en réparti
Sa solution illustre les principales classes d'algorithmes répartis :

- autorisation de tous les sites (messages stables) avec ordonnancement par estampilles [Ricart-Agrawala]
- jeton sans topologie sous-jacente (avec informations d'ordonnancement portées par le jeton) [Suzuki - Kasami]
- jeton sur anneau virtuel
- jeton virtuel matérialisé par un front d'onde
- jeton avec topologie en arbre reconfigurable [Raymond]

Il existe beaucoup d'autres algorithmes - ex : Maekawa ($O(\sqrt{n})$ messages)
En nombre de messages, les algorithmes sur arbres sont les plus efficaces (Raymond, Naïmi-Tréhel) : O(log n) messages

Pour un nombre réduit de sites, l'anneau virtuel est le plus simple

Élection

- **Problème**
  - Parmi un ensemble de processus, en choisir un et un seul (et le faire connaître à tous)
    - sécurité : un seul processus élu
    - vivacité : un processus doit être élu en temps fini
  - L'élection peut être déclenchée par un processus quelconque, éventuellement par plusieurs processus
  - L'identité de l'élu est en général indifférente ; on peut donc (par exemple) choisir le processus qui a le plus grand numéro
  - Difficulté : des processus peuvent tomber en panne pendant l'élection
- **Utilité**
  - Regénération d'un jeton perdu : un jeton et un seul doit être recréé
  - Dans les algorithmes de type "maître-esclave" : élire un nouveau maître en cas de défaillance du maître courant. Exemples (cf plus loin) :
    - validation de transactions
    - diffusion

Election : algorithme de la brute (*bully*) (1)

- **Hypothèses**
  - Le réseau de communication est fiable et synchrone (borne connue sur le temps de communication)
  - Chaque processus connaît l'identité des autres
- **Principe**
  - Les demandes d'élection sont diffusées par inondation
  - Un processus répond à ceux de numéro inférieur au sien
  - Un processus qui ne reçoit aucune réponse constate qu'il est élu
- **Commentaires**
  - Algorithme coûteux en nombre de messages : $O(n^2)$ au pire
  - Résiste bien aux pannes des processus (mais pb choix des délais de garde)

Election : algorithme de la brute (*bully*) (2)

Lancement d'une élection par p_i
demande_elect$_i$ = true
for all j > i : send(p_j, ELECT)
armer_délai_de_garde(T)
// estimation temps max réponse

Réception d'un message ELECT de p_j par p_i
if ¬ demande_elect$_i$: for all k > i : send(p_k, ELECT)
send(p_j, ACK) // on a j < i

Réception du message (ELECTED, j) par p_i
demande_elect$_i$ = false
coordinator$_i$ = j

Initialisation
for all i :
demande_elect$_i$ = false
coordinator$_i$ = max

Déclenchement du délai de garde T par p_i
if (aucun message ACK reçu)
elected = true
send(all, (ELECTED, i))
else armer_délai_de_garde(T1)
// estimation temps max élection

Déclenchement du délai de garde T1 par p_i
if (aucun message ELECTED reçu)
lancer une élection

Réinsertion après panne : un processus qui se réinsère après une défaillance ne doit pas reprendre le numéro qu'il avait avant la défaillance (pourquoi?)

Election sur un anneau (Chang et Roberts)

anneau virtuel unidirectionnel, plusieurs candidats simultanés possibles

Candidature (site p_i):
candidat$_i$ = true
send (succ[i], ELECT, i)

Réception sur p_i du message (ELECT, j)
switch
j > i : send (succ[i], ELECT, j) // j prioritaire
j < i : if not candidat$_i$ // j non prioritaire
candidat$_i$ = true
send (succ[i], ELECT, i)
j = i : send (all, ELECTED, i) // site i élu

O(N log N) messages en moyenne, $\Theta(N^2)$ au pire

Terminaison (1)

Modèle de calcul réparti (rappel)

Ensemble de processus communiquant par messages ; état actif ou passif

Programme (cyclique) de chaque processus :

loop

1. attendre un message (passage temporaire à l'état passif)
2. exécuter un calcul local en réponse à ce message
3. le calcul peut comporter l'envoi de messages à d'autres processus ou la terminaison du processus (passage définitif à l'état passif)

end

Problème de la terminaison

Vérifier que le calcul est achevé

Cela implique deux conditions sur l'état global du système

Tous les processus sont au repos (passifs)

Aucun message n'est en transit

En effet l'arrivée d'un message en transit peut relancer le calcul

Terminaison (2)

Méthodes pour détecter la terminaison

1. **Méthode générale : analyse de l'état global**

   La terminaison est une propriété stable
   On peut donc la détecter par examen d'un état global enregistré (cf Chandy-Lamport)

2. **Méthodes spécifiques : applicables à un schéma particulier de communication**

   Sur un anneau

   Sur un arbre ou un graphe orienté (avec arbre couvrant)

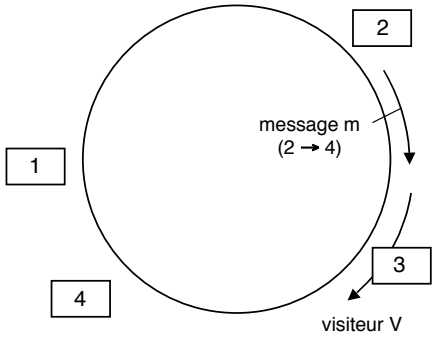
Terminaison sur un anneau

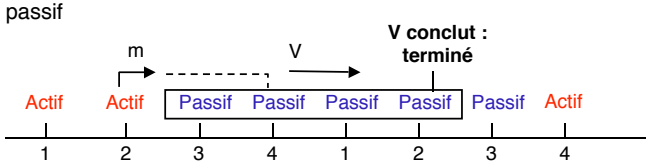
Principe :

visiter l'anneau (dans le sens de la communication) et vérifier que tous les sites sont passifs (après un tour complet)

Difficulté :

un message émis après le passage du visiteur (et non visible par celui-ci) peut venir réactiver (derrière lui) un site trouvé passif





Terminaison sur un anneau (Misra)

Chaque site a une "couleur" blanc ou noir (initialement blanc)
noir = a été actif depuis le dernier passage du jeton, blanc = ¬ noir
Le jeton porte un compteur des sites trouvés passifs
Terminaison = tous les sites blancs après 1 tour

Messages de l'application (sur un site p_i)

NB : les messages et le jeton sont transmis sur l'anneau (FIFO)

attente d'un message ou fin :

```
état = passif ;
```

réception d'un message ou initialisation :

```
état = actif ;
couleur = noir ;
```

Circulation du jeton

réception du jeton (valeur = j) :

```
jeton_présent = true ;
nb = j ;
if ( nb == N and couleur == blanc )
    terminaison détectée
```

émission du jeton :

```
attendre ( jeton_présent and etat==passif )
if couleur == blanc
    envoyer_suivant (jeton, nb + 1)
else
    envoyer_suivant (jeton, 1)
couleur = blanc ;
jeton_présent = false;
```

Terminaison sur un graphe (Dijkstra-Scholten)

Calcul diffusant : lancé à partir d'un nœud origine

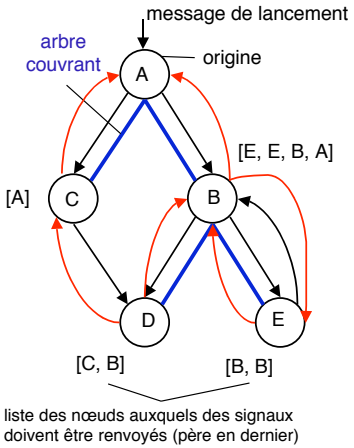
Un nœud qui reçoit un message effectue un calcul et peut soit passer à l'état passif soit envoyer un message à chacun de ses successeurs (le nb total de messages envoyés est fini)

Un arbre couvrant implicite est construit (le premier message reçu par un nœud définit son père dans l'arbre)

Quand un nœud a terminé (feuille) ou reçu un signal de chaque arc sortant, il renvoie un signal sur un de ses arcs entrants (sauf celui du père), pour annuler le "déficit" (nb de messages reçus — nb signaux renvoyés)

Un nœud renvoie un signal à son père quand son déficit a été annulé sur chaque arc sortant (via les signaux reçus sur cet arc) et qu'il a renvoyé un signal sur chaque arc entrant autre que celui issu du père

Le calcul est terminé quand le déficit du nœud origine est nul sur tous ses arcs sortants



E. W. Dijkstra, C. S. Scholten. Termination detection for diffusing computations, *Information Processing Letters*, 11, 1, Aug. 1980, pp. 1-4

Conclusion sur les algorithmes répartis

Rapide parcours de quelques algorithmes de base

Quelques techniques

- Ordonnancement par estampilles (logiques, vectorielles, matricielles)
- Structures de communication (anneau, arbre) imposant un mode de propagation
- Utililisation d'un jeton (rupture de la symétrie)

Propriétés

- Sûreté
- Vivacité (progrès, terminaison)
- Capacité de croissance (évaluation)
- Tolérance aux fautes (reste à examiner plus en détail)