Les systèmes distribués

Objectifs

Concevoir, implémenter et analyser différents protocoles ou algorithmes distribués basés sur le principe de passage de messages entre processus.

Les algorithmes distribués se différentient par :

- La méthode de comunication inter-processus
  - Mémoire partagée (mémoire distribuée, ou distributed shared memory)
  - Diffusion (broadcast)
  - Point à point
- Le modèle temporel (timing model)
  - Synchrone : horloge globale ; l'algorithme distribué se déroule alors sous forme de tours
  - Asynchrone : Les événements sont déclenchés d'une manière asynchrone
  - Partiellement asynchrone : Les horloges sont déclenchées d'une manière synchrone
- Le modèle dans f (?)
  - Pause d'un processus
  - Le processus peut avoir un comportement aléatoire
  - Perte de message
- Un réseau est fiable si :
  - Il n'y a pas de perte de messages
  - Il n'y a pas duplication de messages
  - Le temps de délivrance d'un message est fini

Le problème à résoudre :

- La communication
- Accès à une base de données distribuée
- Application temps réel

Un système distribué est caractérisé par :

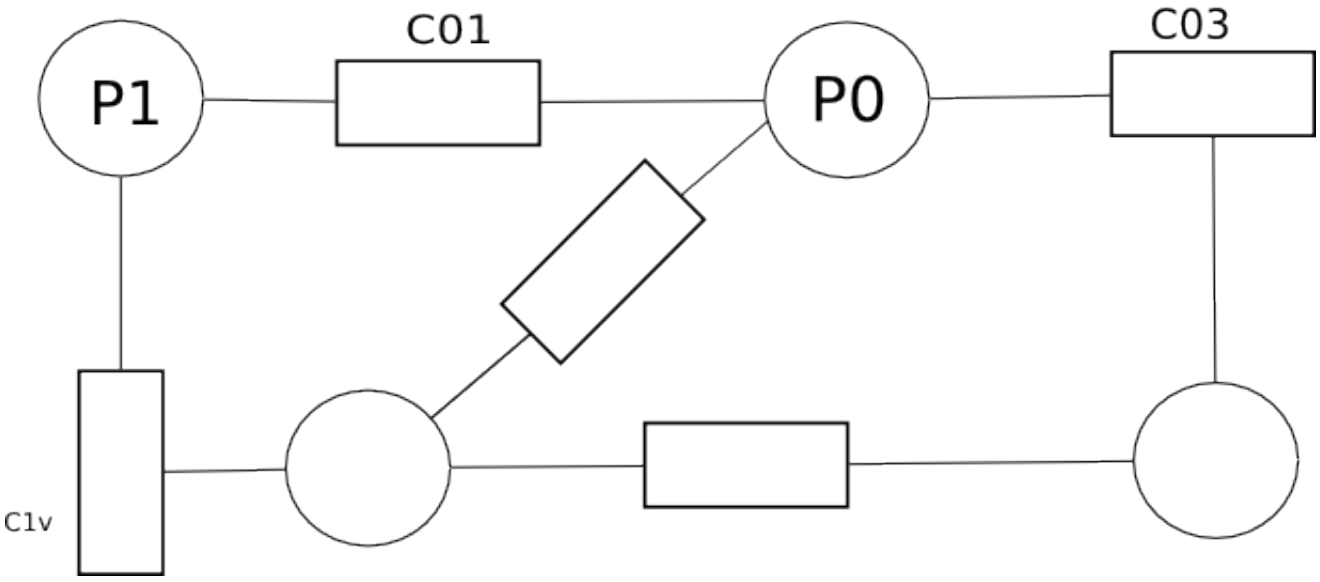
- Son non-déterminisme : On ne peut prédire le déroulement exact de l'algorithme, on ne peut que comprendre certaines propriétés de l'algorithme.
- L'indépendance des activités des processus
- Le nombre de sites comportant le système distribué est inconnu
- La topologie du réseau est inconnue
- La délivrance des messages est incertaine
- Un mécanisme éventuel de tolérance aux pannes

Définitions

Un système distribué est une collection de processus qui échangent des messages via un système de communication.

Le système de communication (S.C.) est constitué de connexions de logiciels. Le comportement des connexions peut être celui de la discipline FIFO (First In First Out).

Le S.C. et le S.D. sont modélisés par un graphe :



Les topologies

La meilleure : un graphe complet $G(V, E)$. Il en existe d'autres :

- En étoile
- Hypercube
- Grille

L'élection distribuée

Soit un système distribué ; l'objectif est d'élire un site particulier pour s'occuper d'une fonction donnée. Ceci permet d'utiliser par exemple le paradigme client/serveur.

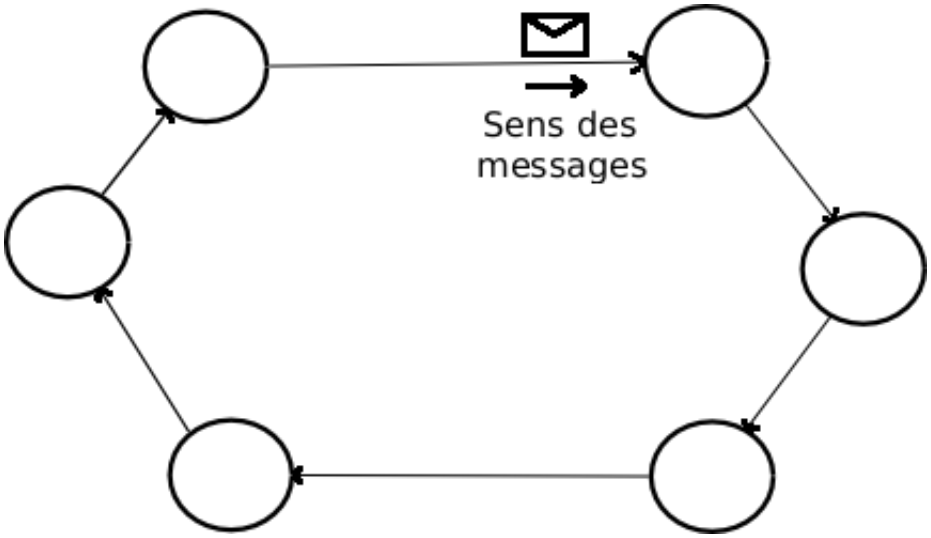
L'élection permet de désigner un serveur. Les messages clients vers le serveur représentent alors des requêtes de service ; des réponses peuvent le cas échéant être envoyées au client.

Nous avons toutefois besoin d'un protocole de négociation permettant de sélectionner un coordinateur.

Un premier algorithme distribué

Hypothèses

On suppose que les sites sont rangés (ou connectés) en anneau unidirectionnel.



Principe

Initialement, un ou plusieurs sites peuvent initier le protocole d'élection en émettant un message et se déclarer participant.

Un site i qui reçoit le message devient participant, ce message contient un identifiant (numérique).

Le site i compare son propre identifiant avec celui du message

- S'il y a égalité, le site i est élu
- S'il est inférieur, il transmet le message à son voisin
- S'il est supérieur, il remplace l'identifiant du message par le sien, et transmet à son voisin

Variables locales

- id_i : L'identifiant unique du site i
- id_c : L'identifiant du coordinateur
- *participe* : Booléen, initialement Faux

Message utilisé

ELECTION : Message envoyé de i vers j conenant un identifiant coordinateur

ELU : Message de fin, indiquant le résultat de l'élection

Code des algorithmes

```
Lors de la réception de INIT par un initiateur Faire
  participe <- Vrai
  envoyer à son voisin de gauche ELECTION(id_i)
Fin


Lors de la réception du message ELECTION(id) par si depuis sj Faire
  Si (id > id_i) Alors
    participe <- Vrai
    envoyer au suivant ELECTION(id)
  Sinon Si (id < id_i) Et (participant = Faux) Alors
    participant <- Vrai
    envoyer à son voisin ELECTION(id_i)
  Sinon Si (id = id_i) Alors
    id_c <- id_i
    envoyer à son voisin ELU(id_c)
  Finsi
Fin


Lors de la réception de ELU(id) Faire
  id_c <- id
  participe <- Faux
  Si (id != id_i) Alors
    envoyer à son voisin ELU(id)
  Finsi
Fin
```

Cas d'un anneau bidirectionnel

Hypothèses

L'anneau est maintenant bidirectionnel ; chaque site ignore le nombre total de sites présents, et les sites sont désignés d'une manière quelconque sur l'anneau.

Nous définirons des primitives adaptées à la communication sur un anneau bidirectionnel :

- envoyerDouble()
- faireSuivre()
- Repondre()

Principe

L'algorithme procède par élections primaires de proche en proche sur des distances croissantes suivant des puissances de 2.

Un site battu se contente de faire suivre les messages d'un coté vers l'autre.

Variables propres à chaque site

- id_i : entier
- *etat* : Flag NONCONCERNE, BATTU, ELU, CANDIDAT, initialisé à NONCONCERNE.
- *lgmax* : entier
- *vainqueur* : entier
- *nbrep* : entier initialisé à 0
- *repOk* : booléen initialisé à vrai

Messages utilisés

- CANDIDATURE(id, lg, lgmax)
- FAIRESUIVRE(bool, id)
- REPONSE(bool, id)

Code de l'algorithme

```
Lors de la réception de INIT par s_i Faire
  etat <- CANDIDAT
  lgmax <- 1
  Tant que (etat = CANDIDAT) Faire
    nbrep <- 0
```

```
    repOk <- Vrai
    envoyerDouble CANDIDATURE(id_i, 0, lgmax)
    attendre(nbrep = 2)
    Si (repOk = Faux) Alors
      etat <- BATTU
    Finsi
    lgmax <- lgmax * 2
  Fin Tantque
Fin


Lors de la réception de REPONSE(bool, id) Faire
  Si (id = id_i) Alors
    nbrep <- nbrep + 1
    repOk <- repOk Et bool
  Sinon
    faireSuivre REPONSE(bool, id)
  Finsi
Fin


Lors de la réception de CANDIDATURE(id, lg, lgmax) Faire
  Si (id < id_i) Alors
    envoyer Reponse(Faux, id)
    Si etat = NONCONCERNE Alors
      [ provoquer une élection ]
    Finsi
  Sinon Si (id > id_i) Alors
    Si (lg < lgmax) Alors
      faireSuivre CANDIDATURE(id, lg, lgmax)
    Sinon
      Reponse(Vrai, id)
    Finsi
  Sinon Si (id_i = id) Alors
    Si (etat != ELU) Alors
      vainqueur <- id_i
      faireSuivre TERMINE(id_i)
    Finsi
  Finsi
Fin
```

```
Lors de la réception de Termine(id) Faire
  Si (vainqueur != id) Alors
    faireSuivre TERMINE(id)
    vainqueur <- id
    etat <- NONCONCERNE
  Finsi
Fin
```

Algorithme d'exclusion mutuelle

Nous allons dorénavant nous intéresser à un algorithme d'exclusion mutuelle sur un anneau avec regénération de jeton (MISTRA ?).

La mise en oeuvre de l'exclusion mutuelle consiste à définir un mécanisme d'arbitrage.

Quelques exemples d'application : Sur le protocole Token-Ring, avoir le jeton est synonyme d'avoir le privilège d'utiliser le média de communication. On utilise aussi des jetons lors d'accès à des ressources critiques, comme un disque dur ou une base de données.

Le jeton est matérialisé par un message spécial qui donne le privilège d'accès à celui qui le détient.

Toutefois, des problèmes importants peuvent se poser :

– Perte du jeton
– Panne de liaison
– Panne d'une machine
– altération du message jeton

La solution consiste à regénérer un jeton. Bien entendu, il faut éviter de regénérer plusieurs jetons (c'est le cas avec des algorithmes distribués qui utilisent le principe de la loi de garde).

Hypothèses

Le système de communication est en anneau. Il y a des possibilités de pannes et d'alteration de messages, mais pas de reprise sur panne.

Principes

Le principe de la solution consiste à utiliser 2 jetons, ping et pong. A chacun des deux jetons est associé une valeur, nbPing et nbPong. Ces deux valeurs sont liées par l'invariant $nbPing + nbPong = 0$, avec initialement $nbPing = 1$ et $nbPong = -1$. Ces deux valeurs comptent le nombre de rencontres des deux jetons.

Variables locales

– m_i : entier permettant de mémoriser le nombre associé au dernier jeton vu par P_i, initialisé à 0.
– *nbPing*, *nbPong* : entiers initialisés à 1 et -1 respectivement.

Message utilisé

JETON(J, nbJ), J étant ping ou pong.

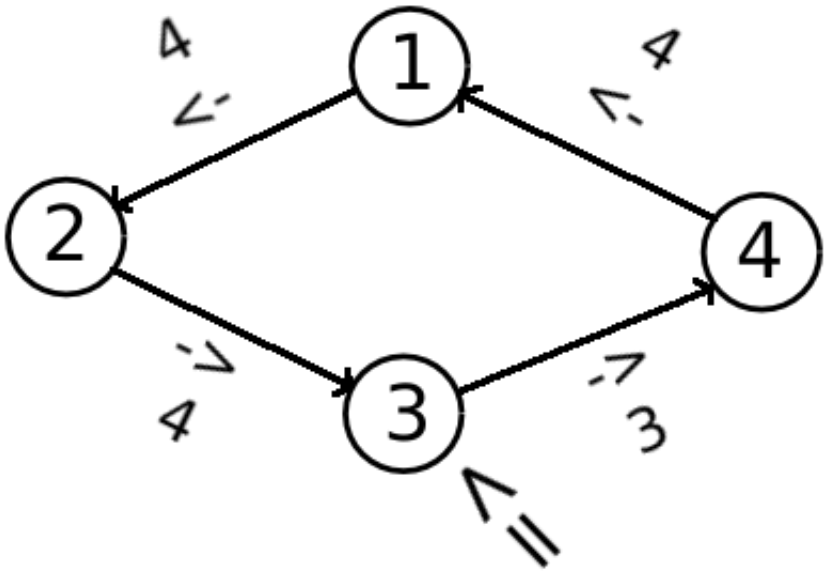
Code des algorithmes

```
Lors de la réception du message JETON(PING, nbPing) Faire
  Si (m_i = nbPing) Alors
    nbPing <- nbPing + 1
    nbPong <- - nbPing
  Sinon
    m_i <- nbPing
  Finsi
  faireSuivre le message au voisin
Fin


Lors de la réception du message JETON(PONG, nbPong) Faire
  Si (m_i = nbPong) Alors
    nbPong <- nbPong - 1
    nbPing <- - nbPong
  Sinon
    m_i <- nbPong
  Finsi
Fin

Lors de la rencontre des 2 jetons Faire
  nbPing <- nbPing + 1
  nbPong <- nbPong - 1
Fin
```

Evaluation du coût de l'algorithme



– Dans le meilleur des cas : n messages
– Dans le pire des cas : $2 \cdot n - 1$

Le coût en terme de nombre N de messages échangés est donc donné par la formule suivante :

$$n \leq N \leq 2 \cdot n - 1$$

Coût de l'algorithme Hirch-Berg et Sinclair

$M = 4 \cdot (2 \cdot n(1 + log(n))) = 2 \cdot n(1 + log(n))$