Séquence 3 - Élection non prédéterminée

Jean.Saquet@info.unicaen.fr

1 Présentation

Ce texte est basé sur l'étude d'algorithmes répartis d'élection : le but est, pour un ensemble de processus ou sites communiquant par messages, de désigner un leader en vue d'un rôle particulier. Les algorithmes dont il est question ont les caractéristiques communes suivantes :

- Les sites ou processus séquentiels communiquent entre eux par messages. Les délais de transmissions de ces derniers sont finis, et ils sont mis dans une file d'attente sur le site destinataire en attendant leur traitement.
- Les lignes de communications, bidirectionnelles, sont fiables : elles ne peuvent ni perdre, ni altérer, ni générer, ni déséquencer les messages. Les processus et les lignes ne peuvent pas tomber en panne.
- Les sites ou processus ont des identités distinctes (nombres entiers pour simplifier), mais chacun ne connaît que les identités de ses voisins immédiats (et la sienne).
- Le graphe constitué par les sites et les lignes de communications est connexe. L'algorithme construit un arbre recouvrant le réseau, dont l'élu est la racine. L'arbre construit est mémorisé localement au moyen de variables Père et Fils sur chaque site.
- Les choix qui doivent être effectués par l'algorithme pour désigner l'élu sont basés sur les identités des sites (les plus grandes identités sont prioritaires).
- L'algorithme d'élection peut être démarré par un nombre quelconque (au moins 1) de sites. Ceux qui n'ont pas "spontanément" démarré l'algorithme seront "éveillés" par la réception d'un premier message relatif à cet algorithme.

Cette dernière caractéristique est importante ; en effet, chaque site du réseau doit pouvoir démarrer l'algorithme, et un nombre quelconque de sites doit pouvoir le démarrer, car sinon (si un site privilégié démarre l'algo), cela revient à avoir déjà un site particulier, et l'élection n'a plus grand sens : le site privilégié peut s'auto-proclamer le chef !

Un algorithme "basique" d'élection a été vu en cours : chaque site peut initier un parcours parallèle du réseau, marqué avec l'identité de l'initiateur. Lorsqu'un site reçoit un message relatif à ce parcours, il le propage s'il n'a pas déjà vu passer un parcours d'identité plus forte, et si sa propre identité n'est pas plus forte. Dans le cas contraire, il stoppe ce parcours (en ne le propageant pas) et, si ce n'était pas déjà fait, lance son propre parcours. Seul le parcours marqué par l'identité de site la plus forte se terminera et son initiateur sera l'élu..

La suite a pour but l'étude d'un autre algorithme d'élection, avec les mêmes caractéristiques listées ci-dessus, mais améliorant celui évoqué ci-dessus sur les points suivants :

- Le travail réalisé par un parcours et une création d'arbre partiel initié par un site qui ne sera finalement pas élu ne sera pas perdu dans la mesure ou cet arbre partiel sera "accroché" à un autre fragment d'arbre contrôlé par un site ayant une priorité plus grande, en vue de la constitution finale de l'arbre couvrant.
- Bien que toujours basé sur la priorité des numéros de sites les plus grands, l'élu ne sera pas nécessairement celui qui a l'identité la plus grande. L'élection ne sera donc pas complètement prédéterminée par les identités des sites.

Ce nouvel algorithme est basé sur les idées suivantes :

Chaque site démarrant spontanément l'algorithme construit un fragment d'arbre ayant pour identité le numéro du site initiateur, et comportant au moins ce dernier. Lorsque deux fragments se "rencontrent", ils fusionnent, celui d'identité plus faible étant "absorbé" par celui d'identité plus forte. Chaque fragment ne comporte à chaque instant qu'un seul site ayant le privilège de pouvoir décider son absorption par un

fragment "plus fort" : au départ c'est l'initiateur de l'algorithme qui porte ce privilège, mais il peut choisir de le transmettre à un autre site du fragment qu'il contrôle. C'est d'ailleurs ce mécanisme qui fait que l'élu final peut ne pas être le site d'identité maximale.

2 Description de l'algorithme :

2.1 états :

Chaque site x considère chacun de ses voisins y qui n'est pas un de ses fils comme étant dans un des états suivants (et le mémorise dans la variable `état[y]`) :
– `candidat` si y est susceptible de faire partie d'un fragment plus fort.
– `actif` si y fait partie d'un segment plus faible
– `inactif` si y appartient au même fragment que x.
Si y est un fils de x, le contenu de `état[y]` est également une de ces trois valeurs, mais avec une sémantique modifiée :
– `candidat` signifie que y possède au moins un voisin candidat
– `actif` signifie que y ne possède pas de voisin candidat, mais a au moins un voisin actif
– `inactif` dans les autres cas.
Si y est le père de x, `état[y]` vaut inactif.

De plus, chaque site se considère comme étant dans l'un des états suivants (règle R1) :
– `ouvert` s'il a au moins un voisin candidat
– `libre` s'il n'a aucun voisin candidat, mais au moins un voisin actif
– `fermé` si tous ses voisins sont inactifs
– `terminé` si pour lui, l'algorithme est terminé.

2.2 Variables de chaque site :

– `Voisins` : constante, ensemble des identités des voisins.
– `ego` : constante portant l'identité du site
– `état` : tableau des états des voisins
– `mon_état` : état du site
– `id_frag` : identité du fragment auquel appartient le site
– `père` : identité du père dans l'arborescence du fragment
– `fils` : ensemble des identités des fils dans l'arborescence du fragment

2.3 Initialisations :

Pour tout voisin i, état[i] := candidat
mon_état := ouvert
id_frag := ego
père := nil
fils := $\varnothing$

2.4 Architecture générale de l'algorithme d'un site :

Tant que mon_état $\neq$ terminé faire A ou B :
– A : si je porte le privilège relatif à mon fragment, et si je ne suis pas en attente de fusion avec un autre fragment, alors < action >
– B : si j'ai au moins un message en attente alors < traitement du premier message >

Le fait de pouvoir, de manière non déterministe, choisir de traiter un message en attente ou, si les conditions sont réunies, prendre une initiative (en l'occurence transmettre le privilège ou tenter une fusion avec un autre fragment) introduit un facteur aléatoire qui fait que le résultat de l'élection n'est pas connu à l'avance.

N.B. : le privilège est mémorisé simplement par le fait que le site privilégié n'a pas de père (cf. ci-dessous).

De manière plus précise, A et B peuvent se détailler ainsi :

```
A : Si père = nil et attente_réponse = faux alors
      Si mon_état = ouvert
            alors si il n'existe pas k ∈ voisins
                        tel que état[k] = candidat et k ∉ Fils
                  alors PASSAGE_JETON
                  sinon TENTE_FUSION
            sinon si mon_état = fermé
                  alors
                        pour tout i ∈ Fils envoyer Fin à i
                        mon_état = terminé
```

les procédures PASSAGE_JETON et TENTE_FUSION étant :

```
PASSAGE_JETON
Choisir k ∈ Fils
      tel que état[k] = candidat
MAJ(k, inactif) {cf. plus bas}
père := k ; Retirer k de Fils
envoyer Jeton(mon_état) à k
```

```
TENTE_FUSION
choisir k voisin
      tel que état[k] = candidat
envoyer à k Conn(id_frag)
attente_réponse := vrai
```

La première de ces procédures est appelée si le site n'a pas de voisin candidat autre que ses fils : il transmet alors à un de ses fils le privilège de pouvoir se "raccrocher" à un autre fragment.

La deuxième est appelée si il reste au moins un voisin susceptible d'appartenir à un fragment plus prioritaire : le site tente alors de fusionner avec ce fragment en émettant un message de connexion (Conn). La variable attente_réponse est là pour empêcher le site de prendre une nouvelle initiative tant qu'il n'a pas reçu la réponse.

```
MAJ (y, état) procédure basée sur la règle R1 vue plus haut
Si état[y] ≠ état
      alors
            état[y] := état
            si il n'existe pas k ∈ voisins, état[k] = candidat
                  alors
                  si il n'existe pas k ∈ voisins, état[k] = actif
                        alors nouv_état := fermé
                        sinon nouv_état := libre
                  sinon nouv_état := ouvert
            Si nouv_état ≠ mon_état alors
                  mon_état := nouv_état
                  si père ≠ nil alors envoyer Maj (nouv_état) à père
```

B : les différents messages susceptibles d'être en attente sont :
- les messages envoyés par une des procédures ci-dessus :
  Jeton (état), Conn (id), Fin
- les réponses à un message Conn :
  Ok (id), Nok, Cousin
- les messages de mise à jour :
  Nrac (id), Maj (état)

En particulier, la réception d'un message Conn (id) donnera lieu à une des trois réponses Ok (avec l'identité du fragment accepteur), Nok ou Cousin selon que le récepteur accepte la fusion (parce que l'id de son propre fragment est supérieure à l'id reçue), la refuse (inégalité stricte dans l'autre sens) ou signale qu'il fait déjà partie du même fragment que l'émetteur.

Le message Maj est envoyé par la procédure MAJ (ne pas les confondre), et le message Nrac (id) sera envoyé par un site s'étant "accroché" avec succès à un autre fragment pour signaler à ses fils qu'ils font désormais partie du fragment d'identité id.

la fonction suivante sera utile à deux reprises :

Fonction transforme_état (état)
Selon le cas
état = ouvert : transforme_état := candidat
état = libre : transforme_état := actif
état = fermé : transforme_état := inactif

le traitement des différents messages est alors le suivant :
- réception de jeton (état) venant de y :
  Fils := Fils ∪ {y} ; Père := nil ; MAJ (y, transforme_état (état)) ;
- réception de Conn (id) venant de y :
  Si id_frag < id alors MAJ (y, candidat) ; envoyer Nok à y ;
  Si id_frag = id alors MAJ (y, inactif) ; envoyer Cousin à y ;
  Si id_frag > id alors
    Fils := Fils ∪ {y} ; MAJ (y, candidat) ;
    envoyer Ok (id_frag) à y
- réception de Ok (id) venant de y :
  attente_réponse := faux ; id_frag := id ;
  Père := y ; MAJ (y, inactif) ;
  Pour tout i ∈ Fils envoyer Nrac (id) à i
- réception de Cousin venant de y :
  attente_réponse := faux ; MAJ (y, inactif) ;
- réception de Nok venant de y :
  attente_réponse := faux ; MAJ (y, actif) ;
- réception de Nrac (id) venant de y :
  id_fragment := id ; Pour tout i ∈ Fils envoyer Nrac (id) à i ;
- réception de Maj (état) venant de y :
  MAJ (y, transforme_état (état)) ;
- réception de Fin venant de y :
  Pour tout i ∈ Fils envoyer Fin à i ;
  mon_état := terminé ;

3 Exemple :

Considérons le réseau comportant 3 sites numérotés 1, 2 et 3, avec deux lignes de communication, une entre 1 et 3 et l'autre entre 1 et 2.

Supposons que le site 1 démarre l'algorithme. Il essaye de se raccrocher au site 3 au moyen d'un message Conn, et celui-ci l'accepte. Le site 1 fait donc partie du fragment commandé par le site 3, et ce dernier est "éveillé" par cette action. Il constate qu'il n'a plus de voisin autre que son fils 1, et décide de lui transmettre le privilège lié au fragment d'identité 3. Le site 1 peut alors essayer de se connecter au site 2. Ce dernier refuse car possède un numéro de fragment (2) inférieur à celui de 1 (3). Les états sont mis à jour sur les sites 1 et 2, et 1 devient "candidat" pour 2. 2 Peut alors décider d'essayer de se connecter à 1, qui accepte puisque l'identité de son fragment est 3. 1 devient donc la racine de l'arbre ayant les deux arêtes du graphe, et son identité est la plus petite dans le réseau.