SDR - Systèmes Distribués et Repartis

Résumé TE2

02 décembre 2025

1 Elections

Le principe d'élection dans un système distribué consiste à choisir un noeud particulier pour assumer un rôle spécifique, souvent celui de coordonnateur ou de leader. Cela est crucial pour assurer la cohérence et la coordination des actions dans le système.

1.1 Aptitude

La notion d'aptitude fait référence à la capacité d'un noeud à remplir efficacement le rôle de leader. Les critères d'aptitude peuvent inclure la puissance de calcul, la disponibilité, la fiabilité, la connectivité du noeud ou d'autres facteurs pertinents.

Cela pourrait servir par exemple à:

- Algorithme à jeton : régénérer un jeton après perte
- Architecture manager/worker : désigner un nouveau manager
- Répartition de charge : sélectionner le serveur le moins chargé

1.2 Propriétés des algorithmes d'élection

- **Sureté**: un seul leader est élu à un moment donné
- **Progrès**: il doit y avoir un élu un jour
- **Validité**: l'élu doit être un noeud valide du système avec la meilleure aptitude
- **Résilience**:
  - Un processus en panne ne doit pas être élu
  - Si le réseau est partitionné, chaque partition doit pouvoir élire un leader

1.3 Algorithme du Bully

1.3.1 Principe

L'algorithme du Bully peut-être résumé par: **j'envoie mon aptitude à tout le monde et tout le monde m'envoie la sienne. Celui qui a l'aptitude la plus élevée gagne**.

1.3.2 Étapes

Dans cet algorithme nous avons 2 étapes principales:

Une élection commence

Je broadcast mon aptitude.

J'attends $2T$ pour tout recevoir.

Je détermine l'élu

Réception d'une aptitude

Si je ne suis pas déjà en élection,

j'entre en élection

> ⚠ **Warning**
>
> Pour que l'algorithme soit correct, il faut que l'on attende $1T$ avant de lancer une nouvelle élection aprés en terminé une.

1.3.3 Performance

- Messages: $O(n^2)$ dans le pire des cas
- Durée d'une élection: Jusqu'à $4T$ dans le pire des cas
  - *$1T$ d'attente, $2T$ de timeout, $1T$ de transit*

1.3.4 Gestion des pannes

Nous allons supposer que: *Un processus ne tombe jamais en panne pendant l'envoi de messages en batch. Soit il envoie tout, soit il n'envoie rien.*

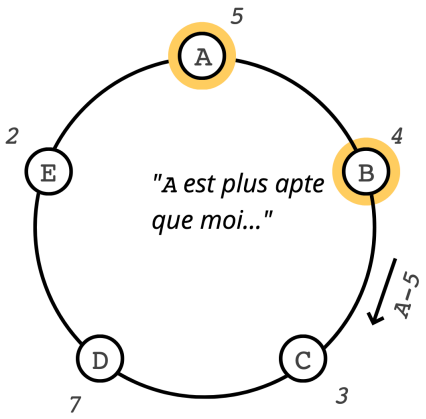
Risque: un noeud qui était en train de gagner tombe en panne.

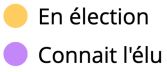
Solution: un détecteur de panne (*un jour*) chez les autres qui relance une élection si le leader tombe en panne.

1.4 Algorithme Chang et Roberts (sans pannes)

L'algorithme de Chang et Roberts peut être résumé par: **j'envoie mon aptitude dans un anneau. Si je reçois une aptitude plus**

grande, je la retransmets sinon j'envoie la mienne. Si je reçois la mienne, j'ai gagné.





1.4.1 Étapes

Si je reçois une aptitude d'un autre
  Je la compare à la mienne
  Je propage la plus grande des deux

Si je reçois ma propre aptitude
  La demande a fait le tour, je suis élu !
  Je propage le résultat

Si je reçois un résultat
  Je le propage si ce n'est pas le mien
  Sinon, l'élection est finie, je suis l'élu

Tour d'annonce

Tour de résultat

> ⚠ **Demandes concurrentes**
>
> Quand on est déjà en élection,
> - Si je reçois une aptitude plus basse, je l'ignore
> - (Sinon, l'algorithme normal s'applique)

1.4.2 Résumé

Demande d'élection
Si je ne suis pas en cours d'élection
- J'entre en élection
- J'envoie mon aptitude et mon ID

Si je suis en cours d'élection
- Je refuse la demande
  (le demandeur réessaiera plus tard)

Réception d'un résultat
Si ce n'est pas moi
- Je note et je propage
- Je sors d'élection

Si c'est moi
- Je suis l'élu
- Je sors d'élection

Réception d'une aptitude
Si je ne suis pas en cours d'élection
- J'entre en élection
- Je compare l'aptitude reçue avec la mienne
- Je propage la plus grande, avec l'ID associé

Si je suis en cours d'élection
- Si c'est la mienne
  - J'envoie le résultat
- Si ce n'est pas la mienne
  - Je compare l'aptitude reçue avec la mienne
  - Je propage celle reçue ssi elle est plus grande.

Pourquoi pas un problème de ne rien envoyer sinon ?

1.4.3 Performances

Communication: $3N$ messages dans le pire des cas (N le nombre de nœuds dans l'anneau).
Durée de l'élection: $O(3NT)$ pour T la durée de transit max

1.5 Algorithme Chang et Roberts (avec pannes)

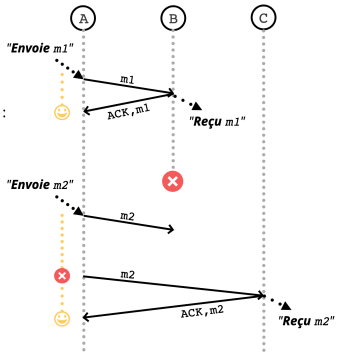
Dans la version avec pannes, nous allons devoir gérer les cas où des nœuds tombent en panne pendant l'élection. Cela se fait au travers d'un mécanisme: **le mainteneur d'anneau**.

1.5.1 Mainteneur d'anneau

Lors d'une demande d'envoi, envoie au prochain processus, et lance un timeout qui, une fois écoulé : envoie le message au prochain dans l'anneau.

Lors d'une réception de ACK, annule le timer associé.

Lors d'une réception de message, notifie la réception de ce message, et envoie ACK avec un id du message.



1.5.2 Résumé

Chaque processus a une estampille t.

Demande d'envoi au nième suivant
- Incrémenter t
- Envoyer message et t au nième suivant dans l'anneau
- Lancer un timer, dont le timeout
  - Demande l'envoi du message au (n+1)ième suivant.

Réception d'un message
- Notifier de la réception
- Répondre `ACK` avec `t_i`

Réception de `ACK` avec `t_i`
- Annuler le timeout associé à `t_i`

1.5.3 Gérer les pannes

Imaginons que le procesus allant gagner l'élection tombe en panne, le message transitant sur le réseau n'ayant que l'ID du gagnant, l'élection ne s'arrêtera jamais et tombera en famine.

Solution: chaque message d'élection contient la liste des nœuds déjà visités. Si un nœud reçoit un message d'élection et qu'il se trouve dans la liste, il sait que le message a fait un tour complet, il peut donc élire le nœud avec la plus

grande aptitude parmi ceux de la liste. Une fois l'élu déterminé, le noeud l'ayant choisi relance un message en transmettant l'id du processus élu et une liste vide ou chaque nœud ajoute son id pour confirmer la réception du message.

> ⚠ **Warning**
>
> Si je reçois le message de validation et que je ne suis pas en élection, je relance l'élection sauf si le résultat correspond déjà à l'élu que j'ai en mémoire.

Election:

```
[
  {id: 3, aptitude: 5},
  {id: 5, aptitude: 8},
  {id: 2, aptitude: 3},
  //etc..
]
```

Validation:

```
{
  elected: 5,
  validators: [3,2,7,9]
}
```

Et si celui qui va être élu tombe en panne après l'envoi d'un résultat ?

- *Il sera élu ! C'est ensuite qu'un détecteur de panne le verra et relancera une élection.*

Et si celui qui va être élu tombe en panne après l'envoi d'une annonce ?

- *Il sera élu ! C'est ensuite qu'un détecteur de panne le verra et relancera une élection.*

Et si celui qui va être élu tombe en panne après l'envoi d'une annonce, mais réapparait avant la réception du résultat ?

- *Il sera élu ! Et personne ne l'aura remarqué !*

Et si on enlevait la supposition qu'un transit dure moins de T unités de temps ?

- *On risque d'élire un processus dont l'aptitude n'est pas la plus grande.*

Demande d'élection par couche applicative
Si je ne suis pas en cours d'élection
- J'entre en élection
- J'envoie une annonce [{moi, apt}]

Si je suis en cours d'élection
- Je refuse la demande

Réception d'une annonce
Si je suis dans la liste de l'annonce
- Je calcule l'élu d'après la liste
- J'envoie un résultat {élu, [moi]}
- Je sors d'élection

Sinon
- J'ajoute {moi, apt} dans la liste
- J'envoie la liste au suivant
- J'entre en élection

Réception d'un résultat
Si je suis dans la liste
- Je n'ai rien à faire !

Si je ne suis pas dans la liste
- Si je ne suis pas en élection et l'élu reçu est différent du mien
  - J'exécute "demande d'élection par couche applicative"
- Sinon
  - Je note le nouvel élu
  - Je m'ajoute à la liste
  - J'envoie un résultat avec l'élu et cette liste
  - Je sors d'élection.

Pourquoi pas plus ?
Je suis dans la liste, tout le monde (pas en panne) a vu le résultat, on peut s'arrêter.

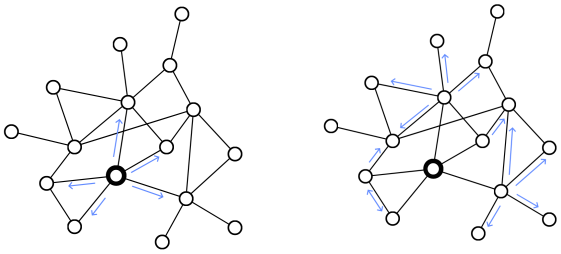
2 Sondes et echos

2.1 Principe

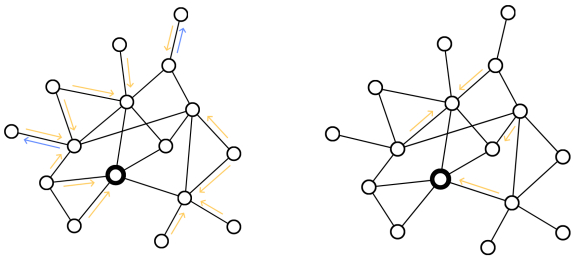
La programmation par sondes et echos est un modèle de communication asynchrone permettant à un processus d'échanger avec tous les autres processus du réseau.

2.2 Phases

Diffusion: Le processus source envoie une sonde à tous ses voisins directs.



Contraction: Chaque processus renvoie un echo au processus qui lui a envoyé la sonde.



2.3 Application sur un arbre

Dans un arbre, pas de risque de boucle car chaque noeud n'a qu'un seul parent.

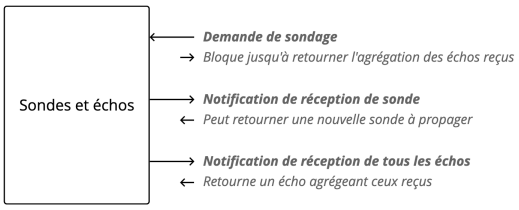
Réception d'une sonde

- Propager aux enfants
- Si pas d'enfants → envoyer echo au parent

Réception d'un echo

- Si tous les echos reçus → envoyer echo au parent
- Si source → terminé

2.4 Interface



2.5 Graphes arbitraires

Risque de boucle → modification de l'algorithme:

Réception d'une sonde

- **Si déjà connue** → arrêter d'attendre l'écho de ce voisin
- Sinon → propager normalement

Réception d'un echo

- Si tous les echos **attendus** reçus → envoyer echo au parent

2.6 Sources multiples

Plusieurs noeuds peuvent initier des sondes simultanément. Chaque sonde doit avoir un **identifiant unique** (combinaison ID source + compteur local, ex: « A-1 », « A-2 »).

2.7 Comportements des noeuds

Source: Envoie sondes aux enfants → attend tous les echos → terminé

Noeud intermédiaire: Attend sonde du parent → propage aux enfants → attend tous les echos → envoie echo au parent

Feuille: Attend sonde du parent → envoie immédiatement echo au parent