SDR - Systèmes Distribués et Repartis

Algorithme Chang et Roberts

18 Novembre 2025

Table des matières

1 Chang et Roberts

1.1 Définition

L'algorithme de Chang et Roberts est un protocole de sélection d'un leader dans un anneau orienté d'ordinateurs. Chaque nœud a un identifiant unique. Le but est que tous les nœuds conviennent d'un seul leader, celui avec l'identifiant le plus élevé.

1.2 Fonctionnement

Le processus A démarre une élection en envoyant un message contenant son identifiant ainsi que son aptitude au noeud suivant dans l'anneau. Chaque nœud qui reçoit un message d'élection compare l'identifiant dans le message avec le sien:

- Si l'identifiant dans le message est plus grand que le sien, il transmet le message au nœud suivant.
- Si l'identifiant dans le message est plus petit que le sien, il remplace l'identifiant dans le message par le sien et transmet le message.

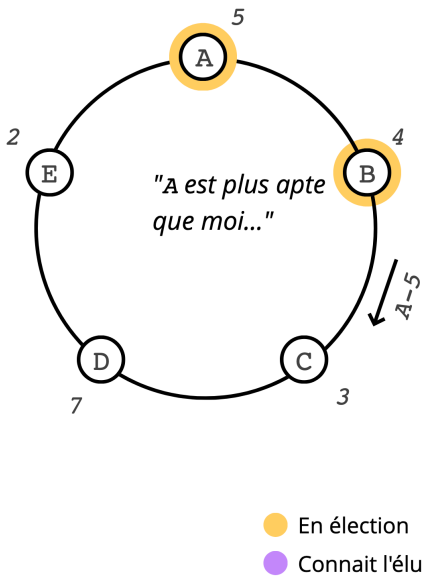

Fig. 1. – Capture des slides du cours – Démarrage d'une élection

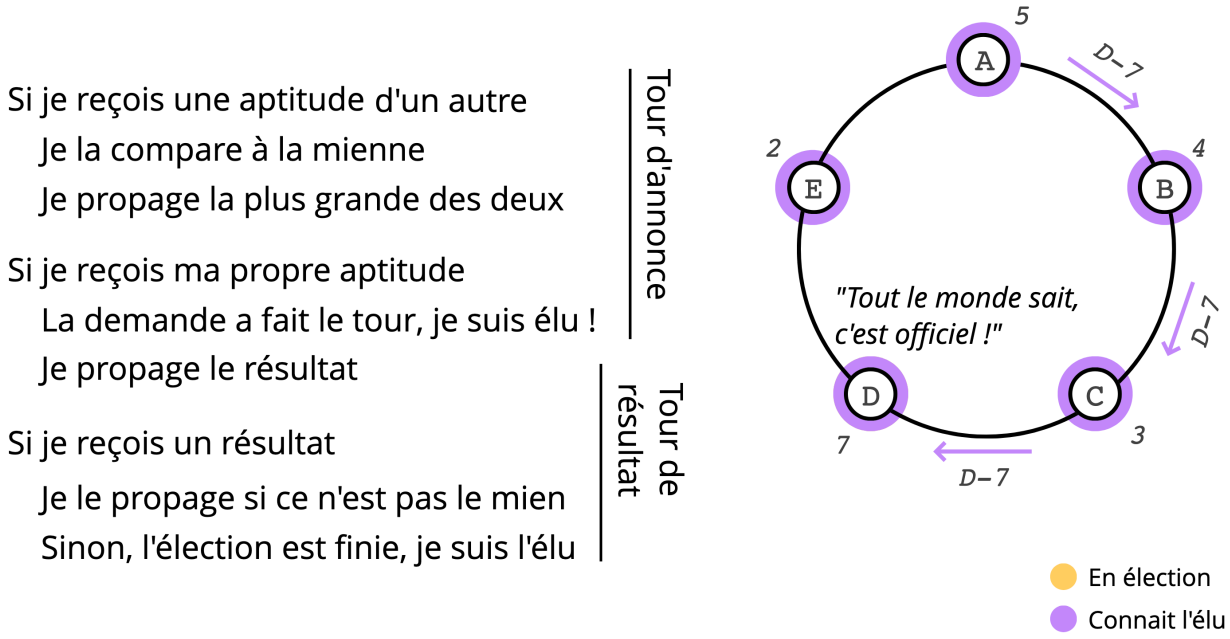

Fig. 2. – Capture des slides du cours – Propagation des messages d'élection

`D` est donc le noeud étant élu comme leader.

1.3 Cas de concurrence

Dans le cas où plusieurs demande d'éléctions sont lancées simultanément, l'algorithme gère cela grâce à une règle simple qui est :

Quand je suis déjà en cours d'élection,

- **Si je reçois une aptitude plus basse, je l'ignore**
- (Sinon l'algorithme continue comme d'habitude)

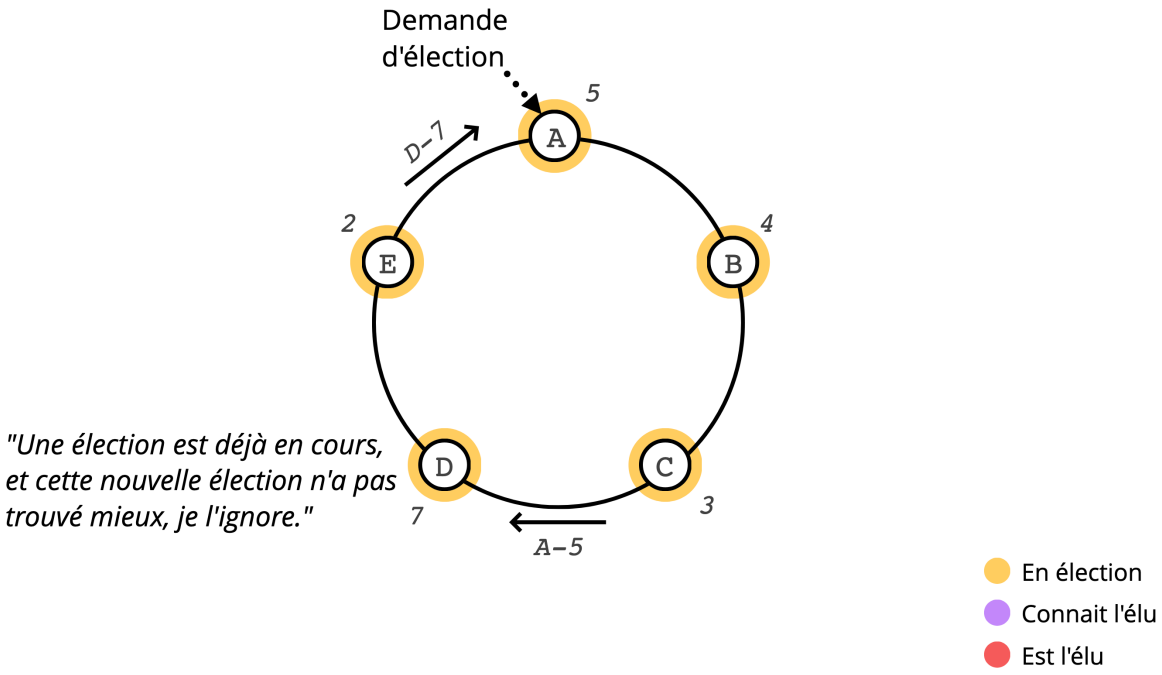

Fig. 4. – Capture des slides du cours – Gestion des élections concurrentes

(B demande une éléction, puis A en demande une autre juste après)

1.4 Résumé

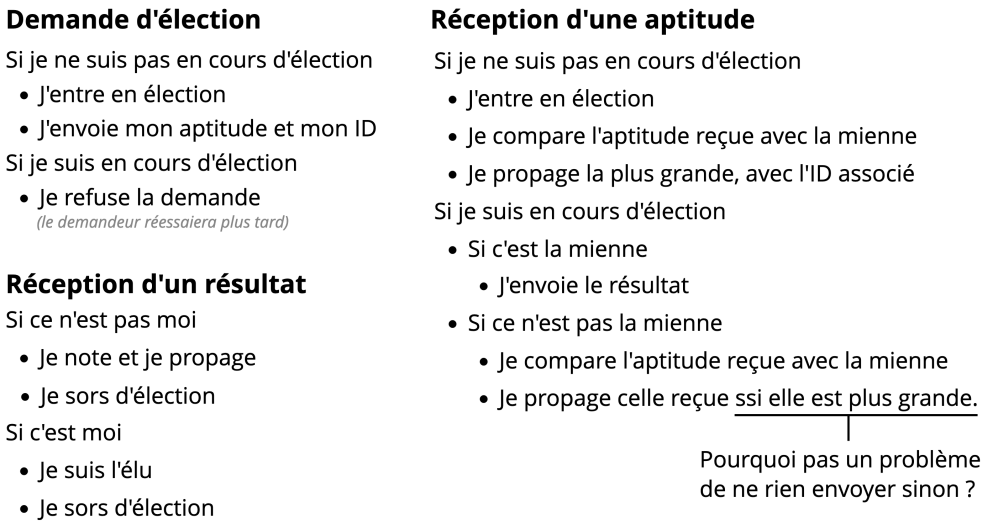

Fig. 5. – Capture des slides du cours – Résumé de l'algorithme de Chang et Roberts

1.5 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

1.6 Pseudo-code

Variables

`self`	*entier, constant*	mon numéro de processus
`apt_self`	*entier*	mon aptitude
`suivant`	*entier, constant*	numéro du suivant dans l'anneau
`inElection`	*booléen, init false*	ssi une élection est en cours
`élu`	*entier*	id du processus élu

Remarques

Plus besoin de stocker les aptitudes de tous les autres.

Plus besoin de dépendre de `T`.

Tous les envois se feront à `suivant`.

Fig. 6. – Capture des slides du cours – Pseudo-code de l'algorithme de Chang et Roberts

Initialisation

Écouter infiniment les événements suivants :

- Demande d'élection
- Demande d'obtention de l'élu par la couche applicative
- Réception d'une annonce `{i, apt_i}`
- Réception d'un résultat `{i}`

Note : les traitements de ces événements sont faits de manière séquentielle.

(On omet la mise à jour de l'aptitude, qui déclenche simplement une élection)

Fig. 7. – Capture des slides du cours – Suite du pseudo-code de l'algorithme de Chang et Roberts

Demande d'élection

```
Si inElection
    Refuser la demande
Sinon
    Envoi d'une annonce {self, apt_self}
    inElection ← true
```

Réception d'un résultat `{i}`

```
élu ← i
inElection ← false
Si élu != self
    Envoi de résultat {i}
```

Fig. 8. – Capture des slides du cours – Fin du pseudo-code de l'algorithme de Chang et Roberts

Demande d'obtention de l'élu

```
Retourner élu.
```

Réception d'une annonce `{i, apt_i}`

```
Si (apt_self, self) > (apt_i, i)
    Si pas inElection
        Demande d'élection
Sinon, si i == self
    Envoi de résultat {i}
    inElection ← false
Sinon
    Envoi d'annonce {i, apt_i}
    inElection ← true
```

Fig. 9. – Capture des slides du cours – Fin du pseudo-code de l'algorithme de Chang et Roberts

2 Chang et Roberts - avec pannes

2.1 Gérer l'anneau avec pannes

Pour gérer ce cas, nous allons ajouter un module `Maintenance d'anneau` qui va permettre de détecter les nœuds défaillants et de réajuster l'anneau en conséquence. L'anneau aura pour rôle de transmettre les messages d'élection et de leader, tandis que le module de maintenance d'anneau s'occupera de la détection des pannes et de la réorganisation de l'anneau.

2.1.1 Suppositions

- Pas de perte de messages.
- Pas de duplication de messages.
- Pas de changement d'ordre de messages.
- Une panne peut être récupérable.
- Durée de transit maximale bornée par une constante T.
- Durées de traitement négligeables.
- Chaque processus connait tous les autres.

2.2 Gestion de l'absence du voisin

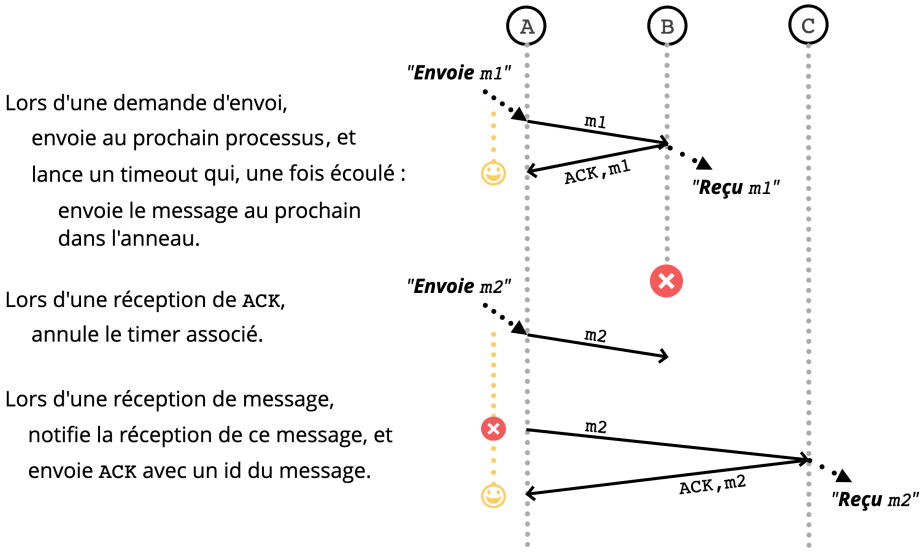

Fig. 10. – Capture des slides du cours – Détection de la panne du nœud B par A

2.3 Résumé

Chaque processus a une estampille `t`.
(pour identifier les messages)

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 de `m` et `t_i`

Notifier de la réception

Répondre ACK avec `t_i`

Réception de ACK avec `t_i`

Annuler le timeout associé à `t_i`

Fig. 11. – Capture des slides du cours – Résumé de l'algorithme de Chang et Roberts avec pannes

2.4 Risque de famine

Dans le cas où le noeud ayant la plus haute aptitude tombe en panne après avoir communiqué son aptitude mais avant d'avoir été élu leader, il y a un risque de famine. En effet, aucun autre nœud n'aura une aptitude plus haute et l'élection ne pourra jamais se terminer et tournera en boucle infinie.

2.5 Solutions au risque de famine

Pour gérer ce risque, nous pouvons approcher le problème en signant le passge du message d'élection. Chaque nœud ajoute sa signature ainsi que son aptitude au message lorsqu'il le transmet. Ainsi, si un nœud reçoit un message d'élection qu'il a déjà signé, il sait que le message a fait un tour complet de l'anneau.

Une fois que le message a fait un tour complet, un nouveau message est transmis en mettant l'id du processus avec la plus haute aptitude et un tableau devant contenir les noeuds validant acceptant le résultat.

> ⚠ **Warning**
>
> - Si je reçois un résultat que j'ai accepté, il a fait le tour et je ne le propage pas.
> - Si je reçois un résultat que je n'ai pas accepté et que je ne suis pas en élection, alors je n'y ai pas participé et je relance l'élection (sauf si mon élu est déjà le bon).

2.6 Résumé

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.

Fig. 12. – Capture des slides du cours – Résumé de l'algorithme de Chang et Roberts avec pannes et gestion du risque de famine

2.7 Pseudo-code

Variables

`self`	*entier, constant*	mon numéro de processus
`apt_self`	*entier*	mon aptitude
`inElection`	*booléen, init false*	ssi une élection est en cours
`élu`	*entier*	id du processus élu

Remarques

Plus besoin de l'id du prochain dans l'anneau : l'envoi est géré par le mainteneur d'anneau.

Fig. 13. – Capture des slides du cours – Pseudo-code de l'algorithme de Chang et Roberts avec pannes et gestion du risque de famine

Demande d'élection

```
Si inElection                                          | Inchangé
    Refuser la demande
Sinon
    Envoi d'une annonce [{self, apt_self}]
    inElection ← true
```

Demande d'obtention de l'élu

```
Retourner élu.                                         | Inchangé
```

Fig. 14. – Capture des slides du cours – Suite du pseudo-code de l'algorithme de Chang et Roberts avec pannes et gestion du risque de famine

Réception d'une annonce de `i` avec liste `participants`

```
Si self apparait dans participants  (L'annonce a fait le tour)
    élu ← i tel que i a l'aptitude max dans participants
    Envoi de résultat {élu, [self]}
    inElection ← false
Sinon
    Ajout de {self, apt_self} dans participants
    Envoi d'annonce avec participants
    inElection ← true
```

Fig. 15. – Capture des slides du cours – Fin du pseudo-code de l'algorithme de Chang et Roberts avec pannes et gestion du risque de famine

Réception d'un résultat de `i` avec `{new_élu, accepted}`

Si `self` n'apparait **pas** dans `accepted` *(Il s'agit d'un résultat inédit)*
- Si pas `inElection` et `élu` n'est pas `new_élu` *(Il s'agit d'une **élection** inédite)*
  - Envoi d'une annonce `[{self, apt_self}]` *(il faut recommencer)*
  - `inElection ← true`
- Sinon
  - `élu ← new_élu` *(j'accepte le résultat)*
  - Ajout de `self` dans `accepted`
  - Envoi d'un résultat `{élu, accepted}`
  - `inElection ← false`

Fig. 16. – Capture des slides du cours – Fin du pseudo-code de l'algorithme de Chang et Roberts avec pannes et gestion du risque de famine