SDR - Systèmes Distribués et Repartis

Algorithme Bully

11 Novembre 2025

Table des matières

1 Problème de l'élection

Dans un groupe de processus, chacun a une aptitude pas nécessairement statique. On souhaite élire un celui qui à la meilleure aptitude.

> **i Info**
>
> Par **aptitude**, on entend une vleur numérique représentant la capacité d'un processus à assumer le rôle de coordinateur. Par exemple, cela peut être basé sur la puissance de calcul, la mémoire disponible, ou toute autre métrique pertinente pour le système distribué en question.

1.1 Dans quel but?

- 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 souhaitées

- Sureté : un seul processus ne peut être élu
- Progrès : il doit y avoir un élu un jour
- Validité : l'élu doit être le participant correct à la plus grande aptitude
- Résilience :
  - Un processus en panne ne doit pas être élu, même s'il redémarre.
  - Si le réseau est scindé en deux, un élu doit exister par partition.

1.2.1 Différences avec une mutex

Mutex	Élection
Essaie d'être équitable	Choisit la meilleure aptitude
Partage l'accès à une section	Partage le droit au titre d'élu
Tout demandeur sera accepté	Un demandeur peut ne jamais être élu
Bloquant si SC est prise	Non-bloquant si un autre est élu

Fig. 1. – Capture des slides du cours – Différence entre élection et mutex

> **💡 Hint**
>
> La ou la mutex s'assure que tout le monde aura droit à son moment, l'élection sert uniquement à décider à qui attribuer un titre spécial d'*élu*.

2 Algorithme Bully

2.1 Idée générale

Tout le monde doit tout savoir.

Chaque processus envoie son aptitude à tous les autres. Chaque processus reçoit donc les aptitudes de tous les autres. Le processus avec la plus grande aptitude s'auto-électe.

2.2 Evénements

Dans cet algorithme, on considère 2 types d'événements :

Deux événements

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.

Fig. 2. – Capture des slides du cours – Événements dans l'algorithme Bully

> ⚠ **Warning**
>
> Si une élection est en cours quand une nouvelle est demandée, elle est différée à la fin de celle en cours.

2.3 Performance

- **Communications** : $O(n^2)$ messages
- **Durée d'une élection** : Jusqu'à $4T$
  - $1T$ d'attente, $2T$ de timeout, $1T$ de transit

2.4 Pseudo-code

Variables

`n`	*entier, constant*	nombre de processus
`self`	*entier, constant*	mon numéro de processus
`apts`	*tableau d'entiers*	aptitudes de tous les processus
`apt_self`	*entier*	mon aptitude
`T`	*entier, constant*	durée maximale d'une transmission
`inElection`	*booléen, init false*	ssi une élection est en cours
`isReqPending`	*booléen, init false*	ssi une élection est en attente
`élu`	*entier*	id du processus élu

Fig. 3. – Capture des slides du cours – Pseudo-code de l'algorithme Bully

Initialisation

Écouter infiniment les événements suivants :
- Demande d'élection par la couche applicative
- Demande d'obtention de l'élu par la couche applicative
- Demande de changement d'aptitude à `new_apt`
- Réception de l'aptitude `apt_i` de `i`
- Timeout

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

Fig. 4. – Capture des slides du cours – Suite du pseudo-code de l'algorithme Bully

Demande d'élection de la couche applicative

Attendre pendant `1T`
Démarrer une élection

Demande d'obtention de l'élu par la couche applicative

Retourner `elu`

Demande de changement d'aptitude à `new_apt`

`apt_self ← new_apt`
Attendre pendant `1T`
Démarrer une élection

Fig. 5. – Capture des slides du cours – Suite du pseudo-code de l'algorithme Bully

Réception de l'aptitude `apt_i` de `i`

```
Si pas inElection
    Démarrer une élection
apts[i] ← apt_i
```

Démarrer une élection

```
Si inElection
    isReqPending ← true
Sinon
    inElection ← true
    apts[j] ← -inf pour tout j
    apts[self] ← apt_self
    Envoi de apt_self à tous les autres processus
    Lancer un timer de 2T
```

Fig. 6. – Capture des slides du cours – Suite du pseudo-code de l'algorithme Bully

Timeout

```
élu ← i tel que apts[i] est le premier maximum de apts.
inElection ← false
Si isReqPending
    isReqPending ← false
    Attendre pendant 1T
    Démarrer une élection
```

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

3 Gestion des pannes

> **i Info**
>
> Nous faisons la supposition qu'un processus ne tombe jamais en panne *pendant* l'envoi de messages en batch. Soit il envoie tout, soit il n'envoie rien.

On suppose alors un *détecteur de panne* chez les autres, surveillant l'élu:

- Si je détecte une panne de l'élu, je lance une nouvelle élection.

Détecteur parfait ou un jour?

- Un jour suffit : si suspicion annulée, relancer une élection.
- En cas de faux positif, le même élu sera à nouveau choisit.