# Théorie des applications réparties

## Algorithmique répartie

# En général on ne peut pas résoudre les problèmes répartis

## Un problème simple (problème des deux armées)

Deux armées distantes doivent attaquer en même temps, comment faire ?

## Formalisation

A, B, messages entre A et B, pertes et délais possibles, $(q_A, q_B)$ état du système, l'objectif est de passer directement de $(q_A^0, q_B^0)$ à $(q_A^1, q_B^1)$

## Exercice

Proposer un protocole que devraient suivre les deux armées pour résoudre leur problème

# En général on ne peut pas résoudre les problèmes répartis

## Un problème simple (problème des deux armées)

Deux armées distantes doivent attaquer en même temps, comment faire ?

## Formalisation

A, B, messages entre A et B, pertes et délais possibles, $(q_A, q_B)$ état du système, l'objectif est de passer directement de $(q_A^0, q_B^0)$ à $(q_A^1, q_B^1)$

## Exercice

Proposer un protocole que devraient suivre les deux armées pour résoudre leur problème

## Solution

C'est impossible en général... (pourquoi ? et les probabilités ?)

# Le cadre de ce cours

## Agents

On considère un ensemble de $n$ agents qui ne peuvent pas tomber en panne ni tricher (dans un premier temps)

## Communications

Les agents communiquent au moyen de messages qui ne peuvent pas se perdre, ni être dupliqués, ni être réordonnés

## Algorithme réparti

On appel algorithme réparti un algorithme :

- qui sera exécuté par chaque agent,
- qui ne sera pas différencié selon les agents

# Plan du cours

# Le problème du consensus réparti

## Le problème

Plusieurs agents proposent chacun une valeur et doivent chacun décider une valeur, l'objectif est que tous les agents décident la même valeur

## Propriétés attendues

- **Validité :** la valeur décidée par chaque agent qui décide est l'une des valeurs proposées
- **Accord :** tous les agents qui décident décident la même valeur
- **Terminaison :** tous les agents décident en temps fini

# Le problème du consensus réparti

## Le problème

Plusieurs agents proposent chacun une valeur et doivent chacun décider une valeur, l'objectif est que tous les agents décident la même valeur

## Propriétés attendues

- **Validité :** la valeur décidée par chaque agent qui décide est l'une des valeurs proposées
- **Accord :** tous les agents qui décident décident la même valeur
- **Terminaison :** tous les agents décident en temps fini

## Solution(s)

Des idées ? Et si certains agents ont des pannes ?

# Un algorithme de consensus très simple

## Pour chaque agent

1. Envoyer à tout le monde sa valeur initiale
2. Recevoir $n$ valeurs (on reçoit sa propre valeur)
3. Décider la plus petite des valeurs reçues

## Preuve

- **Validité :** trivial
- **Accord :** pas de perte de messages + pas de pannes
- **Terminaison :** pas de perte de messages + pas de pannes

# Un algorithme de consensus très simple

## Pour chaque agent

1. Envoyer à tout le monde sa valeur initiale
2. Recevoir $n$ valeurs (on reçoit sa propre valeur)
3. Décider la plus petite des valeurs reçues

## Preuve

- **Validité :** trivial
- **Accord :** pas de perte de messages + pas de pannes
- **Terminaison :** pas de perte de messages + pas de pannes

**Mais...** s'il y a un agent qui tombe en panne, que peut-il se passer ?

# Plan du cours

# Crash

## Définition

Un agent qui crashe s'arrête de fonctionner définitivement (plus d'envoi de messages, plus de changements d'état local, etc)

## Remarques

- Parfois on considère le cas où les processus qui crashent peuvent redémarrer (pas dans ce cours)
- Consensus uniforme : tous les processus qui décident décident la même valeur (même s'ils crashent ensuite)

# Crash

## Définition

Un agent qui crashe s'arrête de fonctionner définitivement (plus d'envoi de messages, plus de changements d'état local, etc)

## Remarques

- Parfois on considère le cas où les processus qui crashent peuvent redémarrer (pas dans ce cours)
- Consensus uniforme : tous les processus qui décident décident la même valeur (même s'ils crashent ensuite)

**Question :** pouvez-vous imaginer une panne pire qu'un crash ?

# Les généraux byzantins

## Le problème

Une armée assiège une ville. Le général et ses lieutenants commandent chacun une partie des troupes. Ils doivent s'accorder sur un plan de bataille commun. Parmi eux il y a des traîtres.

**Question :** comment s'assurer que les généraux loyaux arrivent à s'accorder sur un plan ?

Pouvez-vous proposer des solutions ?

# Les généraux byzantins

## Le problème

Une armée assiège une ville. Le général et ses lieutenants commandent chacun une partie des troupes. Ils doivent s'accorder sur un plan de bataille commun. Parmi eux il y a des traîtres.

**Question :** comment s'assurer que les généraux loyaux arrivent à s'accorder sur un plan ?

Pouvez-vous proposer des solutions ?

## Solutions

- Messages oraux (synchrones) : pas plus de 1/3 de traîtres pour résoudre le problème
- Messages écrits non falsifiables : nombre quelconque de traîtres

# Pannes byzantines

## Définition

Un agent byzantin est un agent qui ne collabore pas nécessairement avec les autres : il peut faire n'importe quoi

## Par exemple

- Envoyer de fausses informations
- Envoyer des informations différentes à des agents différents
- Retarder les messages
- ...

# Pannes byzantines

## Définition

Un agent byzantin est un agent qui ne collabore pas nécessairement avec les autres : il peut faire n'importe quoi

## Par exemple

- Envoyer de fausses informations
- Envoyer des informations différentes à des agents différents
- Retarder les messages
- ...

## En pratique

On considère un agent byzantin comme malicieux : il fait toujours la pire chose possible du point de vue des agents non byzantins

# Plan du cours

# Pannes et consensus

## Un résultat fameux (cas asynchrone)

Il suffit qu'un seul agent puisse subir un crash pour qu'il ne soit pas possible de garantir qu'un algorithme de consensus fonctionnera bien

## Pourquoi ?

Comment différencier un agent qui a subit un crash d'un agent dont les messages mettent très longtemps à parvenir aux autres ?

# Pannes et consensus

## Un résultat fameux (cas asynchrone)

Il suffit qu'un seul agent puisse subir un crash pour qu'il ne soit pas possible de garantir qu'un algorithme de consensus fonctionnera bien

## Pourquoi ?

Comment différencier un agent qui a subit un crash d'un agent dont les messages mettent très longtemps à parvenir aux autres ?

## Des solutions ?

Considérer un cadre un peu moins général :

- synchrone (les messages arrivent en temps borné),
- détecteurs de fautes (on peut s'assurer qu'un agent est bien en panne),
- validité/terminaison avec une forte probabilité

# Consensus synchrone avec crash

## Du point de vue d'un agent ($t$ pannes au maximum, $t < n/2$)

1. Initialiser $est$, $r \leftarrow 0$
2. Envoyer à tout le monde $PHASE_1(r, est)$
3. Attendre d'avoir reçu les messages $PHASE_1(r, est')$ de $n - t$ agents
4. Si toutes les valeurs $est'$ reçues sont égales à une valeur $v$, envoyer $PHASE_2(r, v)$ à tout le monde, sinon envoyer $PHASE_2(r, ?)$ à tout le monde
5. Attendre d'avoir reçu les messages $PHASE_2(r, est'')$ de $n - t$ agents
6. Si
   - toutes les valeurs $est''$ reçues sont égales à $v'$, décider $v'$
   - toutes les valeurs $est''$ reçues sont égales à ?, $est \leftarrow est'$ pour $est'$ la plus petite des valeurs reçues en phase 1
   - les valeurs $est''$ reçues sont toutes soit $v'$ soit ?, $est \leftarrow v'$
7. $r \leftarrow r + 1$
8. Retourner à l'étape 2

# Consensus avec crash et détecteur de fautes

## Du point de vue d'un agent ($t$ pannes au maximum, $t < n/2$)

1. Initialiser $est$, $r \leftarrow 0$
2. $est \leftarrow f(r, est)$ //On utilise un détecteur de fautes pour implanter $f$
3. Envoyer à tout le monde $PHASE_1(r, est)$
4. Attendre d'avoir reçu les messages $PHASE_1(r, est')$ de $n - t$ agents
5. Si toutes les valeurs $est'$ reçues sont égales à une valeur $v$, envoyer $PHASE_2(r, v)$ à tout le monde, sinon envoyer $PHASE_2(r, ?)$ à tout le monde
6. Attendre d'avoir reçu les messages $PHASE_2(r, est'')$ de $n - t$ agents
7. Si
   - toutes les valeurs $est''$ reçues sont égales à $v'$, décider $v'$
   - toutes les valeurs $est''$ reçues sont égales à ?, ~~$est \leftarrow est'$ pour $est'$ la plus petite des valeurs reçues en phase 1~~
   - les valeurs $est''$ reçues sont toutes soit $v'$ soit ?, $est \leftarrow v'$
8. $r \leftarrow r + 1$
9. Retourner à l'étape 2

# Consensus avec crash et détecteur de fautes, suite

## Hypothèse supplémentaire

Les agents sont numérotés de 1 à $n$, chacun connaît son numéro

## Implantation de $f(r, est)$ à l'aide d'un détecteur de fautes

1. $c \leftarrow (r \mod n) + 1$ //Coordinateur
2. Si $c = i$ envoyer $F(r, est)$ à tout le monde //l'agent $i$ appelle $f$
3. Attendre d'avoir reçu le message $F(r, est')$ ou de savoir que l'agent $c$ est en panne
4. Si le message a été reçu, retourner $est'$ Si le message n'a pas été reçu, retourner $est$

## À propos du détecteur de fautes

Il suffit de pouvoir garantir que, après la panne d'un processus, on peut être au courant en temps fini (et il n'est pas possible de résoudre le consensus asynchrone avec un détecteur de fautes plus faible)

# Consensus probabiliste avec crash

## Du point de vue d'un agent ($t$ pannes au maximum, $t < n/2$)

1. Initialiser $est$, $r \leftarrow 0$
2. Envoyer à tout le monde $PHASE_1(r, est)$
3. Attendre d'avoir reçu les messages $PHASE_1(r, est')$ de $n - t$ agents
4. Si toutes les valeurs $est'$ reçues sont égales à une valeur $v$, envoyer $PHASE_2(r, v)$ à tout le monde, sinon envoyer $PHASE_2(r, ?)$ à tout le monde
5. Attendre d'avoir reçu les messages $PHASE_2(r, est'')$ de $n - t$ agents
6. Si
   - toutes les valeurs $est''$ reçues sont égales à $v'$, décider $v'$
   - toutes les valeurs $est''$ reçues sont égales à ?, $est \leftarrow est'$ pour $est'$ prise au hasard parmi les valeurs reçues en phase 1
   - les valeurs $est''$ reçues sont toutes soit $v'$ soit ?, $est \leftarrow v'$
7. $r \leftarrow r + 1$
8. Retourner à l'étape 2

# Consensus probabiliste avec agents byzantins

## Du point de vue d'un agent ($t$ pannes au maximum, $t < n/5$)

1. Initialiser $est$, $r \leftarrow 0$
2. Envoyer à tout le monde $PHASE_1(r, est)$
3. Attendre d'avoir reçu les messages $PHASE_1(r, est')$ de $n - t$ agents
4. Si au moins $n - 2t$ valeurs $est'$ reçues sont égales à une valeur $v$, envoyer $PHASE_2(r, v)$ à tout le monde, sinon envoyer $PHASE_2(r, ?)$ à tout le monde
5. Attendre d'avoir reçu les messages $PHASE_2(r, est'')$ de $n - t$ agents
6. Si
   - au moins $n - 2t$ valeurs $est''$ reçues sont égales à $v'$, décider $v'$
   - au moins $n - 2t$ les valeurs $est''$ reçues sont égales à ?, $est \leftarrow est'$ pour $est'$ prise au hasard parmi les valeurs reçues en phase 1
   - au moins $n - 2t$ valeurs $est''$ reçues sont soit $v'$ soit ?, $est \leftarrow v'$
7. $r \leftarrow r + 1$
8. Retourner à l'étape 2

# Consensus probabiliste avec agents byzantins

## Du point de vue d'un agent ($t$ pannes au maximum, $t < n/5$)

1. Initialiser $est$, $r \leftarrow 0$
2. Envoyer à tout le monde $PHASE_1(r, est)$
3. Attendre d'avoir reçu les messages $PHASE_1(r, est')$ de $n - t$ agents
4. Si au moins $n - 2t$ valeurs $est'$ reçues sont égales à une valeur $v$, envoyer $PHASE_2(r, v)$ à tout le monde, sinon envoyer $PHASE_2(r, ?)$ à tout le monde
5. Attendre d'avoir reçu les messages $PHASE_2(r, est'')$ de $n - t$ agents
6. Si
   - au moins $n - 2t$ valeurs $est''$ reçues sont égales à $v'$, décider $v'$
   - au moins $n - 2t$ les valeurs $est''$ reçues sont égales à ?, $est \leftarrow est'$ pour $est'$ prise au hasard parmi les valeurs reçues en phase 1
   - au moins $n - 2t$ valeurs $est''$ reçues sont soit $v'$ soit ?, $est \leftarrow v'$
7. $r \leftarrow r + 1$
8. Retourner à l'étape 2

**Rappel :** on ne peut pas réaliser le consensus en présence de plus de $n/3$ agents

# Plan du cours

# Élection (*leader election*)

## Le problème

Plusieurs agents doivent élire un leader parmi eux

## Propriétés attendues

- **Unicité :** un unique agent (pas en panne) se considère leader
- **Accord :** les autres agents qui ont décidé de leur rôle considèrent que l'agent précédent est le leader
- **Terminaison :** tous les agents décident de leur rôle en temps fini

## Solution(s)

Comment utiliser un algorithme de consensus pour réaliser une élection ?

# Élection (*leader election*)

## Le problème

Plusieurs agents doivent élire un leader parmi eux

## Propriétés attendues

- **Unicité :** un unique agent (pas en panne) se considère leader
- **Accord :** les autres agents qui ont décidé de leur rôle considèrent que l'agent précédent est le leader
- **Terminaison :** tous les agents décident de leur rôle en temps fini

## Solution(s)

Comment utiliser un algorithme de consensus pour réaliser une élection ?

## Remarque

Élire un leader peut servir de base pour réaliser le consensus avec crash

# Réception de messages en ordre total

## Rappel : réception causale de messages (*causal broadcast*)

Comment s'assurer que si un message $m_1$ est reçu par un agent $c_i$ avant l'émission par $c_i$ d'un message $m_2$, alors $m_1$ sera reçu par tout agent $c_j$ avant $m_2$ ?

## Réception de messages en ordre total (*total order broadcast*)

Tous les messages doivent être reçus dans le même ordre par tous les agents

# Réception de messages en ordre total

## Rappel : réception causale de messages (*causal broadcast*)

Comment s'assurer que si un message $m_1$ est reçu par un agent $c_i$ avant l'émission par $c_i$ d'un message $m_2$, alors $m_1$ sera reçu par tout agent $c_j$ avant $m_2$ ?

## Réception de messages en ordre total (*total order broadcast*)

Tous les messages doivent être reçus dans le même ordre par tous les agents

## Équivalence avec le consensus

- **Dans un sens :** consensus pour décider du prochain message à recevoir
- **Dans l'autre sens :** le premier message reçu donne la valeur à décider pour réaliser le consensus