

Plan

- 1 Le consensus
 - Définition
 - Modèles, défaillances
 - Universalité, impossibilité
- 2 Système synchrone
 - Sans défaillance
 - Défaillance d'arrêt
 - Défaillance byzantine
- 3 Système asynchrone
 - Sans défaillance
 - Défaillance d'arrêt
 - Détecteur de défaillances



Consensus en asynchrone

Résultat d'impossibilité (FLP85)

Le consensus est impossible à réaliser dans un système asynchrone où un seul processus peut subir une défaillance d'arrêt.

→

- Refuser (ou ignorer) les défaillances : *best effort*
- Affaiblir le problème : *k*-consensus
- Affaiblir le problème : terminaison probabiliste
- Renforcer le système : détecteur de défaillance

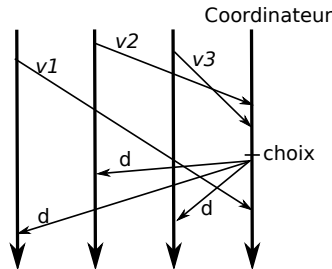


Consensus avec coordinateur



Hypothèse : système asynchrone fiable, pas de défaillance

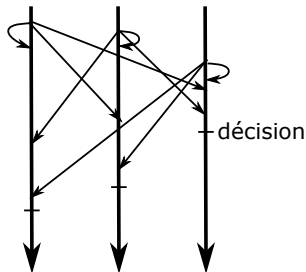
- Chaque processus envoie sa valeur à un coordinateur désigné à l'avance
- Au bout d'un certain temps (après avoir reçu au moins une valeur), le coordinateur choisit une valeur d
- Le coordinateur envoie d à **tous** les processus (diffusion fiable)
- Chaque processus décide d
- \Rightarrow tous les processus décident identiquement en temps fini **non borné**



Consensus symétrique



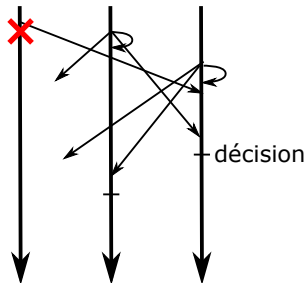
- Chaque processus diffuse sa valeur à tous
- Quand un processus a reçu **toutes** les valeurs, il applique un algorithme déterministe (p.e. min) pour choisir la valeur de décision
- Tous les processus utilisent le même algorithme
- \Rightarrow tous les processus décident identiquement en temps fini **non borné**



k -consensus avec arrêt de processus



- Chaque processus envoie sa valeur à tous les autres
- Quand un processus a reçu (au moins) $n - (k - 1)$ valeurs, il décide le min.



Si $f < k < n$ alors :

- Terminaison en temps fini (non borné) pour les processus corrects
- Au plus f valeurs non reçues par tous \Rightarrow au plus $f + 1$ décisions distinctes



Algorithme à terminaison probabiliste



Algorithme à terminaison probabiliste, assurant la sûreté (accord, intégrité, validité) si $f < \frac{n}{2}$ (f = nb de fautes, n = nb de sites).

- Tours asynchrones
- A chaque tour, le processus :
 - Rapporte sa valeur courante en la diffusant à tous
 - Attend $n - f$ rapports
 - Si une valeur obtient une majorité absolue ($> \frac{n}{2}$) alors propose cette valeur en la diffusant à tous sinon propose “?”
 - Attend $n - f$ propositions
 - Si une valeur est reçue au moins $f + 1$ alors décide cette valeur et termine
 - Si une proposition non “?” a été reçue alors prendre cette valeur sinon prendre **au hasard** 0 ou 1

1. *Another advantage of free choice : Completely asynchronous agreement protocols*, Michael Ben-Or, Principles of Distributed Computing, 1983



Algorithme de Ben-Or

```
Processus  $P_i(v_i)$ ,  $0 \leq i < n$   
   $x_i \leftarrow v_i$  // estimation courante de  $p_i$ , ( $v_i \in \{0, 1\}$ )  
   $k_i \leftarrow 0$  //  $k$  est le n° de tour  
  boucle  
     $k_i \leftarrow k_i + 1$   
    envoyer Report( $k_i, x_i$ ) à tous  
    attendre  $n-f$  messages Report( $k_i, *$ ) // "*"  $\in \{0, 1\}$   
    si reçu plus de  $n/2$  Report( $k_i, v$ ) avec le même  $v$   
      envoyer Proposal( $k_i, v$ ) à tous  
    sinon  
      envoyer Proposal( $k_i, ?$ ) à tous  
      attendre  $n-f$  messages Proposal( $k_i, *$ ) // "*"  $\in \{0, 1, ?\}$   
      si reçu au moins  $f+1$  Proposal( $k_i, v$ ) avec le même  $v$  alors  
        décider  $v$  et terminer  
      si reçu au moins un Proposal( $k_i, v$ ) alors  $x_i \leftarrow v$   
      sinon  $x_i \leftarrow 0$  ou  $1$  aléatoirement  
  finboucle
```

Algorithme de Ben-Or : preuve

Deux processus p_i et p_j ne peuvent pas proposer deux valeurs distinctes dans le même tour.

Si p_i propose 0, il a reçu $> \frac{n}{2}$ rapports égaux à 0. Donc p_j n'a pas pu recevoir $> \frac{n}{2}$ rapports égaux à 1 (et inversement pour 1/0).

Si un processus p_i décide v à un tour k , tous les processus p_j démarreront le tour $k + 1$ avec $x_j = v$.

Si p_i décide v , c'est qu'il a reçu $f + 1$ propositions pour v . Au même tour, p_j a reçu $n - f$ propositions, donc au moins une était v (intersection non vide des propositions reçues). D'après le résultat précédent, il n'a pas pu recevoir d'autre proposition non-?
→ il fixe son x_j à v .

Algorithme de Ben-Or : preuve (2)

Si tous les processus (non crashés) ont la même valeur au début d'un tour, tous décident cette valeur dans ce tour.

Si tous les processus non crashés ont la même valeur v , chaque processus rapporte v aux autres. Comme $n - f > n/2$, chacun propose v . Comme $n - f \geq f + 1$, chacun décide v .



Algorithme de Ben-Or : sûreté

Accord

Tous les processus (qui décident) décident de la même valeur.

Découle directement des lemmes précédents.

Validité

La valeur décidée est l'une des valeurs proposées.

Supposons que p décide v qui n'était pas proposé initialement. Donc tous les processus avaient $1 - v$ initialement. Donc tous avaient la même valeur $1 - v$, et tous, y compris p , décident de cette valeur (lemme précédent). Contradiction.

Intégrité

Un processus ne décide qu'une seule fois.

Le processus termine après avoir décidé.

Algorithme de Ben-Or : vivacité

Terminaison

Tout processus correct finit par décider avec probabilité 1.

- 1 Nous disons qu'une valeur v est k -fixée si tous les processus non crashés possède v au tour k (et donc vont décider v au plus au tour $k + 1$)
- 2 À un tour quelconque, une valeur v a une probabilité $\geq (\frac{1}{2})^n$ d'être fixée (un processus ne fixe pas nécessairement x_i par tirage aléatoire, d'où le \geq).
- 3 Au tour k , $Prob[\text{aucune valeur n'est } k\text{-fixée}] < 1 - (\frac{1}{2})^n$
- 4 $Prob[\text{aucune valeur n'est fixée pour les } k \text{ premiers rounds}] < (1 - (\frac{1}{2})^n)^k$
- 5 $Prob[\text{une valeur est } k\text{-fixée pendant les } k \text{ premiers rounds}] \geq 1 - (1 - (\frac{1}{2})^n)^k$
- 6 Converge vers 1 quand k croît.

Décteur de défaillances : Motivation



Résultat d'impossibilité (FLP85)

Le consensus est impossible à réaliser dans un système asynchrone où un seul processus peut subir une défaillance d'arrêt.

- Le consensus en asynchrone est impossible car on ne sait pas distinguer un processus défaillant (arrêté) d'un processus lent.
- On va **supposer** l'existence d'un détecteur de défaillances, qui indique si un processus est fautif ou non.
- FLP \Rightarrow un détecteur parfait est impossible.
- Il existe des détecteurs **imparfaits** (i.e. qui peuvent se tromper) qui suffisent pour réaliser le consensus !
- FLP \Rightarrow ces détecteurs imparfaits sont impossibles mais on peut en construire des **approximations réalistes**.



Décteur de défaillances



Définition

- Un détecteur de défaillances est un service réparti composé de détecteurs locaux à chaque processus (site).
- Un détecteur fournit à son processus local une liste des processus qu'il **suspecte** d'être défaillants.
- Les détecteurs locaux coopèrent (ou pas) pour établir cette liste.
- Propriétés :
 - Complétude : peut-on ne pas suspecter un processus défaillant ?
 - Exactitude : peut-on suspecter un processus correct ?
- Équivalent à un **oracle** (éventuellement imparfait)

1. *Unreliable Failure Detectors for Reliable Distributed Systems*, Tushar Chandra and Sam Toueg. Journal of the ACM. March 1996.



Complétude



Complétude (*completeness*)

- **Complétude forte** : tout processus défaillant finit par être suspecté par **tout** processus correct
- **Complétude faible** : tout processus défaillant finit par être suspecté par **un** processus correct

Complétude faible et complétude forte sont équivalentes :

- En complétude faible, tout processus défaillant finit par être détecté par au moins un processus
- Périodiquement, chaque processus diffuse sa liste de processus suspectés
- Alors tous les processus finiront par obtenir l'information de suspicion = complétude forte
- (hypothèse : la diffusion est fiable, ce qui est réalisable en asynchrone avec défaillance d'arrêt)



Exactitude permanente



Peut-on suspecter à tort un processus ?

Exactitude permanente (*accuracy*)

- **Exactitude forte** : **aucun** processus n'est suspecté avant qu'il ne devienne effectivement défaillant.
- **Exactitude faible** : il existe **un** processus correct qui n'est jamais suspecté par aucun autre processus.

Exactitude forte : en particulier, un processus correct ne peut jamais être suspecté par aucun autre processus.



Exactitude inévitable



Exactitude après une période initiale de chaos

Exactitude inévitable

- **Exactitude finalement forte** : au bout d'un certain temps, aucun processus n'est suspecté avant qu'il ne devienne défaillant.
- **Exactitude finalement faible** : au bout d'un certain temps, il existe un processus correct qui n'est plus jamais suspecté par aucun autre processus

Exactitude finalement forte : équivalent à : au bout d'un certain temps, aucun processus correct n'est suspecté par aucun autre processus correct

(Finalement = inévitablement = *eventually*)

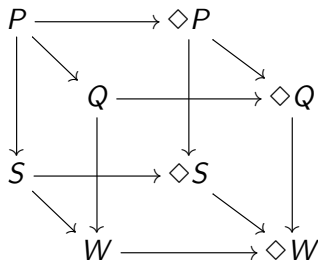


Classes de détecteurs de défaillances



	Exactitude			
	forte	faible	finalement forte	finalement faible
Complétude forte	P	S	$\diamond P$	$\diamond S$
Complétude faible	Q	W	$\diamond Q$	$\diamond W$

P = perfect, S = strong, W = weak



Consensus avec détecteur parfait P

Données

- f = nombre de défaillances d'arrêt à tolérer
- pour chaque processus p_i , un vecteur V_i contenant les valeurs proposées par les autres processus et connues de p_i
- $V_i[j]$ = valeur proposée par p_j telle que connue par p_i
- Initialement $V_i[i] = v_i$ (valeur proposée par p_i) et $V_i[j] = \perp (j \neq i)$



Consensus avec détecteur parfait : principe



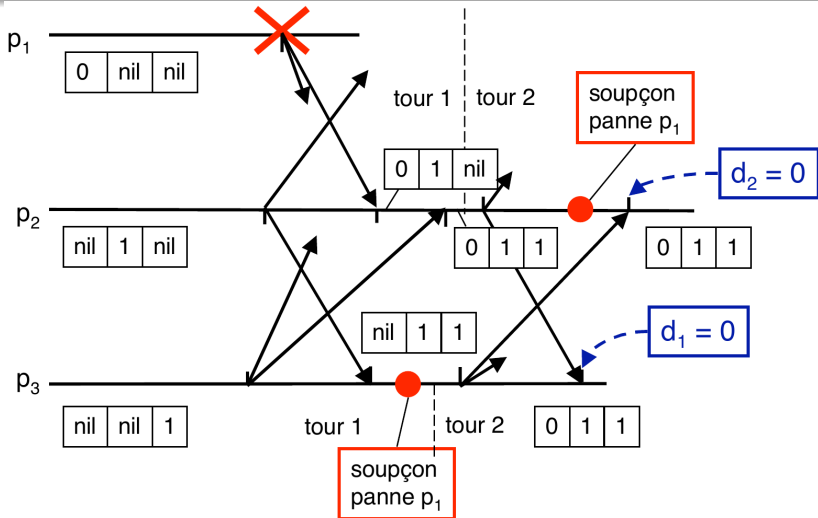
- $f + 1$ tours asynchrones : à chaque tour :
 - chaque processus envoie aux autres son vecteur V_i
 - chaque processus met à jour son vecteur V_i avec les nouvelles valeurs apprises (celles telles que $V_i[j] = \perp$)
 - quand un processus a reçu un message de tous les processus non suspectés par son détecteur P , il passe au tour suivant.
- Après $f + 1$ tours, le processus décide la première valeur non \perp de son vecteur.

Remarques :

- il suffit de transmettre les nouvelles valeurs apprises au tour précédent \Rightarrow un processus n'envoie une valeur qu'une seule fois.
- $V_i[j]$ est stable : mis à jour au plus une fois et plus jamais modifié si non \perp .



Consensus avec détecteur parfait : exemple



Consensus avec détecteur parfait : sûreté – accord

La valeur décidée est la même pour tous les processus corrects.

Supposons que p_i et p_j décident deux valeurs différentes \Rightarrow au tour $f + 1$, V_i et V_j sont différents. Soit x une valeur de V_i non présente dans V_j .

- x a été transmis à p_i par un processus correct p_k , mais p_j n'a pas attendu ce message car il suspectait p_k .

Impossible : **exactitude du détecteur parfait**.

- x a été transmis à p_i au **dernier** tour par un processus p_k qui s'est arrêté avant de transmettre x à p_j (si pas au dernier, p_i aurait transmis la valeur à p_j).

p_k connaissait cette valeur depuis un tour t . Nécessairement, $t =$ le tour précédent (f), car sinon p_j aurait attendu et reçu un message de p_k avec la valeur x (p_k correct aux tours $< f + 1$).

Par récurrence, puisque p_j ne connaît pas x , le processus qui devait le lui envoyer s'est chaque fois arrêté avant.

\Rightarrow **$f + 1$ défaillances** : impossible



Consensus avec détecteur parfait : sûreté – intégrité, validité

- *Intégrité : tout processus décide au plus une fois.*
Unique point de décision après $f + 1$ tours.
- *Validité : la valeur décidée est l'une des valeurs proposées.*
Les vecteurs V_i sont constitués de valeurs connues par les autres sites, et au départ il n'y a que les $V_i[i]$ qui contiennent les valeurs proposées (pas d'invention de valeurs).



Consensus avec détecteur parfait : vivacité – terminaison

Terminaison : tout processus correct décide au bout d'un temps fini

Supposons qu'un processus p_i ne décide pas. Alors il est bloqué dans un tour en attente d'un message provenant d'un processus

p_j :

- p_j est correct. Alors son message finira par arriver (en temps fini non borné) ;
- p_j est défaillant. Alors il finira par être suspecté (**complétude du détecteur parfait**) et p_i ne l'attendra plus.



Consensus avec détecteur S



On sait qu'un processus correct n'est jamais suspecté. Faire n tours pour être sûr de le voir.

- $n - 1$ tours identiques à l'algorithme précédent.

Au $n - 1$ -ième tour, chaque processus a un vecteur, mais tous les vecteurs des processus corrects ne sont pas nécessairement identiques car des processus corrects ont pu être suspectés.

- tour supplémentaire : chaque processus diffuse son vecteur et attend un vecteur de chacun des autres processus qu'il ne suspecte pas. Les \perp reçus effacent de son vecteur les valeurs non \perp .

À la fin, tous les processus corrects ont le même vecteur : des valeurs proposées par des processus corrects peuvent ne pas y être (processus suspectés à tort) mais au moins la valeur du processus correct jamais suspecté est présente.

- décision comme précédemment.



Consensus avec détecteur $\diamond S$

Pour tolérer f défaillances, il faut $2f + 1$ processus.

Principe :

- Un coordinateur tournant. Chaque processus sert de coordinateur à tour de rôle.
- Exactitude finalement faible : il existe un moment où un processus correct sera à la fois non suspecté et coordinateur.
- Un coordinateur (non suspecté) qui réunit une majorité de processus corrects (possible car $n \geq 2f + 1$) peut décider d'une valeur.
- Il diffuse alors cette valeur qui est retenue par tous les processus corrects.



Synthèse

détecteur	nombre de défaillances	nombre de tours
P	$n - 1$	$f + 1$
S	$n - 1$	n
$\diamond S, \diamond W$	$\frac{n-1}{2}$	fini non borné

Plus faible détecteur de défaillances

Le détecteur de défaillances $\diamond W$ (complétude faible, exactitude finalement faible) est le plus faible détecteur de défaillances permettant de résoudre le consensus en asynchrone avec défaillance d'arrêt.

1. *The Weakest Failure Detector for Solving Consensus*. Tushar Chandra, Vassos Hadzilacos and Sam Toueg. Journal of the ACM. July 1996.



Implantation des détecteurs de défaillances



Les détecteurs de défaillances P , S , $\diamond S$, $\diamond W$ ne sont pas réalisables en asynchrone avec défaillances d'arrêt (FLP), mais ils sont réalisables en **supposant assez** de synchronisme.
On se donne δ = délai maximal de transmission d'un message.

Détecteur actif (*ping*)

Périodiquement, p envoie un message à q et attend un acquittement. En absence de réponse après 2δ , p suspecte q .

Détecteur passif (*heartbeat*)

Périodiquement (période T), q diffuse un message "je suis vivant". Si à $T + \delta$ après le message précédent, le processus p n'a rien reçu, il suspecte q .

Nécessite des horloges synchronisées, réalisables sous l'hypothèse synchrone d'un délai maximal de communication.



Implantation des détecteurs de défaillances

Estimation de δ :

- Trop petite : fausses détections
- Trop grande : temps trop long avant détection

Si le système est effectivement asynchrone, il n'existe pas de δ qui évite les fausses détections, et on ne peut pas garantir l'exactitude même faible (tous supposés défectueux par au moins un autre), sauf à tuer les processus suspectés. . .



Détecteur de défaillances – bilan

- Apport théorique : identifier le minimum nécessaire pour réaliser le consensus ; cadre unique de comparaison
- Apport pratique : isoler la résolution d'un problème réparti et la gestion de la détection des défaillances



Conclusion sur le consensus

Universalité

Consensus → tout objet (ayant une spécification séquentielle)

Communication par messages, synchrone

- Consensus réalisable même avec défaillances complexes (mais alors très coûteux)
- Modèle peu réaliste

Communication par messages, asynchrone

- Consensus impossible même avec défaillance simple
- Contournements :
 - détecteurs de défaillances
 - hypothèse de synchronisme suffisamment longtemps (Paxos)