IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Cohérences

Fabien Dagnat

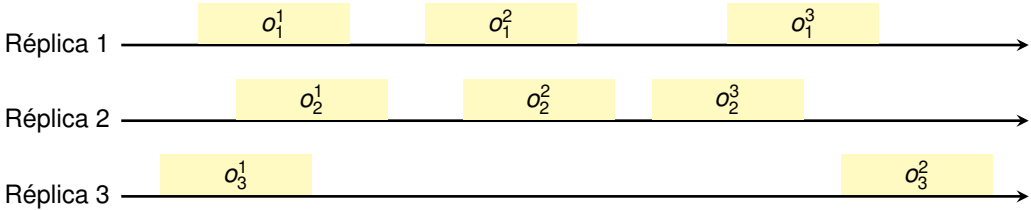
ILSD – FIAB – 6– 2025-2026

Le problème

- La distribution implique des données *répliquées* sur plusieurs serveurs
  - tolérance aux pannes d'un serveur
  - accès plus efficace à des données (elles sont plus proches)
- Mais cela implique des risques d'incohérences entre les réplicas
  - temps de propagation des mises à jours
  - partition du réseau
  - écritures multiples « simultanées »

$\Rightarrow$ On parle de cohérence

Contexte



- Les opérations ont des interférences (par exemple portent sur la même donnée)
- Pour simplifier par la suite, on se limite à des lectures et écritures sur des variables mais les notions sont plus générales

Différentes formes de cohérences

1. Stricte [*strict*]
2. Forte (Linéarisable) [*strong*]
3. Séquentielle [*sequential*]
4. Causale [*causal*]
- À terme [*eventual*]

Hiérachie de la plus forte à la plus souple

Avancement

Cohérence stricte

Définition

Toute lecture d'une variable retourne la valeur écrite le plus récemment

- S'il y a plusieurs réplicas, il faut un temps global
  - « plus récemment » doit être non ambiguë
  - connaissance exacte du moment de l'opération
  - faisable uniquement dans un mono-processeur mono-cœur où chaque opération est en exclusion mutuelle
  - le système est souvent bloqué, ce qui est contradictoire avec la disponibilité

Cohérence forte

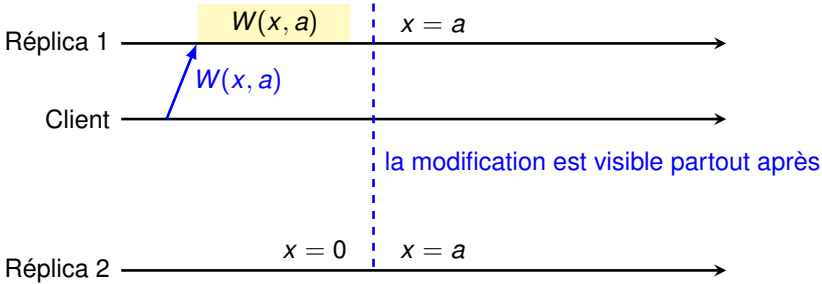
Définition

Toutes les opérations sont ordonnées par un ordre commun à tous les réplicas ; toute opération qui se termine avant le début d'une autre doit la précéder dans cet ordre ; le résultat d'une lecture correspond à la dernière écriture suivant cet ordre

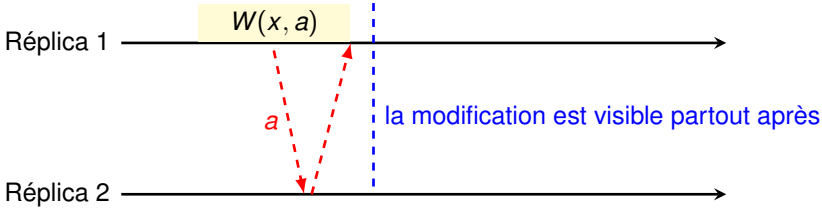
Linéarisabilité

- Chaque opération parait appliquée à un instant t (plus de durée)
- Il y a une forme d'exigence de temps réel, l'opération semble se manifester partout en même temps

Cohérence forte : exemple

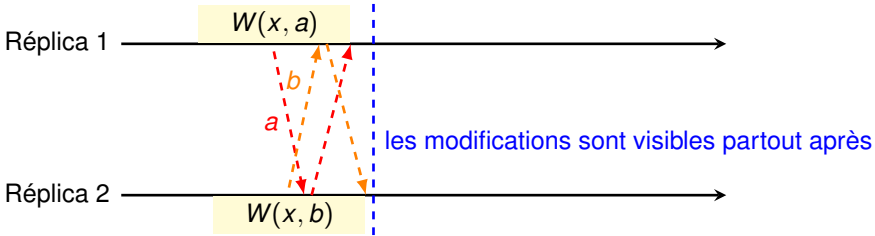


Cohérence forte : implémentation (1/4)



- Tous les réplicas doivent avoir acquitté l'écriture (au minimum diffusion fiable)
- Cohérence forte (linéarisabilité) et faible latence sont incompatibles

Cohérence forte : implémentation (2/4)



- Tous les réplicas doivent avoir acquitté l'écriture et s'être mis d'accord sur un ordre (consensus)
- Le résultats des écritures apparaît partout dès la validation

Cohérence forte : implémentation (3/4)

Deux grandes stratégies

- Identification d'un nœud primaire
  - centralisation des opérations (et du calcul de l'ordre) sur le nœud primaire
  - il diffuse les copies aux autres serveurs
  - algorithme : 2PC (Two-Phase commit)
- Plusieurs nœuds primaires
  - écriture/lecture plus accessibles/locales
  - contenu et ordre des réplicas calculés avant le retour de l'opération
  - Algorithme : consensus

Cohérence forte : conclusion

Cohérence forte et disponibilité sont incompatibles

Cohérence séquentielle

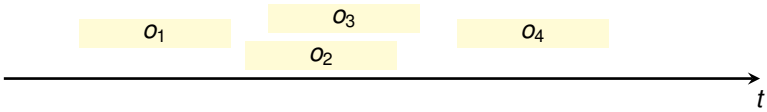
Définition

Toutes les opérations sur une variable apparaissent dans le même ordre pour tous les réplicas

- Pas de référence à un moment plus récent
- Pas besoin de temps global
- Mais on construit un *ordre total commun*

Cohérence séquentielle : implantation

- On s'appuie sur l'ordre défini par la causalité
  - utilisation d'une horloge vectorielle
  - chaque opération est datée
  - si $t(o_1) < t(o_2)$ alors $o_1 \rightarrow o_2$
- S'il sont concurrents ($t(o_1)$ et $t(o_2)$ non comparables) on choisit un ordre ; le même pour tous



	Valide	Valide	Non valide
Réplica 1 :	$o_1 \rightarrow o_2 \rightarrow o_3 \rightarrow o_4$	$o_1 \rightarrow o_3 \rightarrow o_2 \rightarrow o_4$	$o_1 \rightarrow o_2 \rightarrow o_3 \rightarrow o_4$
Réplica 2 :	$o_1 \rightarrow o_2 \rightarrow o_3 \rightarrow o_4$	$o_1 \rightarrow o_3 \rightarrow o_2 \rightarrow o_4$	$o_1 \rightarrow o_3 \rightarrow o_2 \rightarrow o_4$

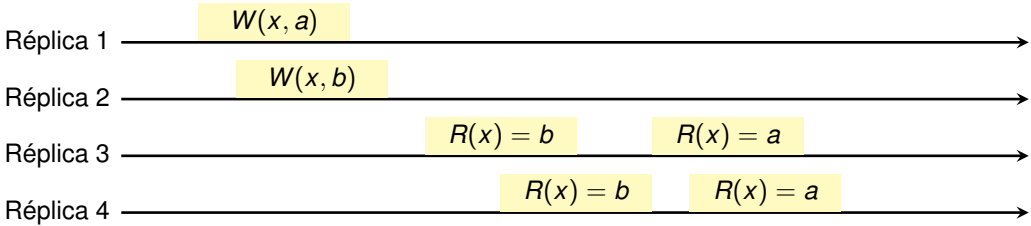
Cohérence causale

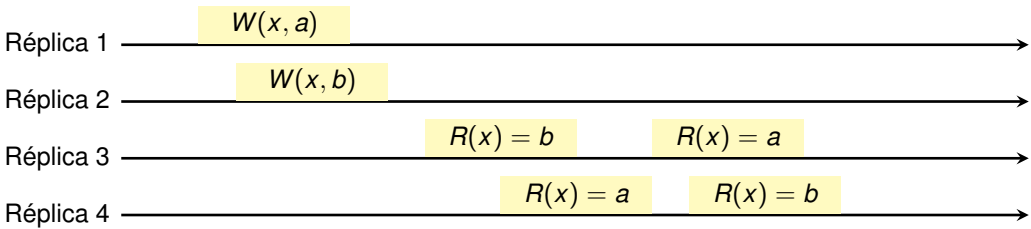
Définition

Toutes les opérations causales sur une variable apparaissent dans le même ordre pour tous les réplicas

- On relache le choix de l'odre global
- Les réplicas sont libres d'ordonner les opérations non causales comme ils veulent

Cohérence séquentielle versus causale





Cohérence à terme

Définition

Les opérations sur les réplicas ne sont pas coordonnées ; la convergence des réplicas est garantie par des propriétés mathématiques s'il n'y a plus d'opérations de modification

- On accepte temporairement des « incohérences »
- La disponibilité est primordiale
- Mises à jour concurrentes possibles
- Il faut gérer les anomalies que les incohérences provoquent

$\Rightarrow$ *Conflict-free Replicated Data Type* (CRDT)

Cohérence à terme : usage

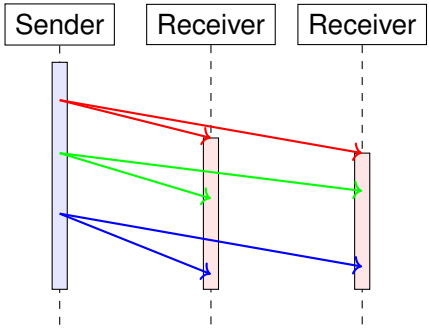
- Particulièrement adaptée pour des données avec peu de conflit de mise à jour
  - ex 1 : des données qui changent peu
  - ex 2 : un seul réplica fait les écritures
- Ainsi, les seuls conflits sont des lectures de données obsolètes
- Le lecteur peut se contenter de ces données
- Par exemple : DNS, contenu de site web (répliqués)
- Les mises à jour se propagent et au final tous les réplicas ont la *bonne* donnée

Cohérence à terme : principes

- Les opérations qui commutent peuvent être concurrentes
  - add(e) ; rm(f) $\equiv$ rm(f) ; add(e) $\equiv$ add(e) || rm(f)
- Sinon
  - sérialisabilité : add(e) || rm(e) $\equiv$ add(e) ; rm(e) ou rm(e) ; add(e)
  - pas de perte de mise à jour
  - le résultat ne dépend pas de l'ordre ; choisir une sémantique
    - last write wins : $t(add(e)) < t(rm(e)) \implies e \notin S \vee t(rm(e)) < t(add(e)) \implies e \in S$
    - error : $\perp_e \in S$
    - add wins : $e \in S$
    - rm wins : $e \notin S$

Avancement

Diffusion



Quelles propriétés peut-on avoir ?

- délais de transmission ? borné, fini
- pas de perte de message
- équité : garantie que tous reçoivent
- atomicité : tous ou personne
- les messages ne se doublent pas
- même ordre de réception pour tous
- les messages sont délivrés dans l'ordre causal

Même en cas de panne

- Les propriétés du canal de communication utilisé sont des hypothèses
- Celles du service de diffusion, l'objectif (la spécification)

Quelques principes sur la transmission

- Les messages ne sont pas transmis instantanément ; ils peuvent être perdu
- Il est impossible de distinguer une perte de message d'un délais très long de transmission
- La notion de temps global n'a pas de sens – chaque nœud possède son propre temps ; il est impossible de dater de manière absolue un événement
- La notion de causalité doit être reconstruite ; horloge de Lamport, horloge vectorielle, etc

Erreurs [Rappel]

Il faut prendre en compte le fait qu'un acteur (processus, programme, machine, canal, réseau, humain, etc.) peut faire des erreurs.

- Par omission ; oublie d'envoyer un message, de répondre, ...
- Arbitraires ; envoie d'un mauvais message (volontairement, on dit qu'il est malicieux ou Byzantin)

Pour les omissions, le modèle le plus simple consiste à considérer qu'un acteur tombe en panne (crash-stop model) ; quand il omet d'envoyer un message, il omet d'envoyer tous les suivants...

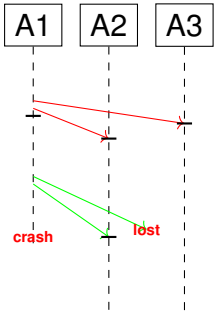
Hypothèses sur les canaux

- *Perfect (or Reliable) links* (PL)
  - (Validity) Si p_i et p_j sont corrects, alors tout message émis par p_i vers p_j est finalement délivré à p_j
  - (No duplication) Aucun message n'est délivré plus d'une fois
  - (No creation) Aucun message n'est délivré sauf à avoir été émis
- *Reliable FIFO links* (FIFO)
  - Perfect links
  - (FIFO) Les messages sont délivrés dans le même ordre que leur émission

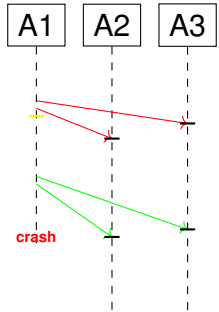
Propriétés de la diffusion

- Différentes familles
  - *Best-effort* : validité, Pas de duplication, pas de création (comme PL)
  - *Reliable* = BE + *Agreement* : si un message *m* est délivré à un destinataire *correct*, alors tous les destinataires corrects recevront le message
  - *Uniform* = BE + *Uniform agreement* : pour tout message *m*, si un destinataire reçoit *m* alors tous les destinataires corrects reçoivent *m*
- De nombreux algorithmes
- Peut respecter la causalité ou non !

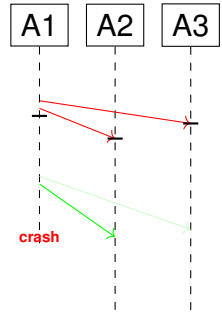
Best-Effort vs Reliable Broadcast



bien que correct A3 ne reçoit pas un message que A2 a reçu

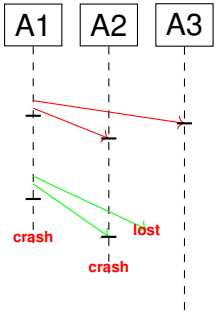


tous les corrects reçoivent le message

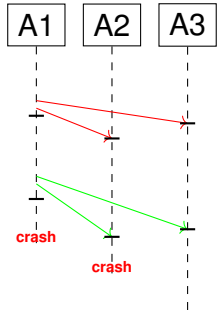


ou aucun

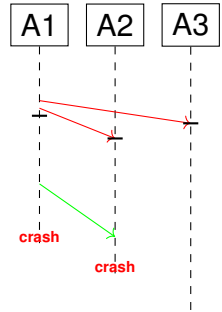
Reliable vs Uniform Broadcast



diffusion fiable, car A2 non correct, $m2$ peut être perdu



même si un récepteur crash, c'est tout le monde...



... ou personne

Les abstractions de diffusion fiable

- *Best-effort broadcast* : garanti la fiabilité *si l'émetteur est correct*
- *Reliable broadcast* : garanti la fiabilité *même si l'émetteur est incorrect*
- *Uniform reliable broadcast* : prend en compte *les récepteurs incorrects*
- *Total reliable broadcast* : reliable broadcast *avec même ordre de réception*
- *Causal reliable broadcast* : reliable broadcast *avec respect de la causalité*

Une façon de réaliser un ordre total est de s'appuyer sur un algorithme de consensus

Avancement

CRDT : 3 familles

- Commutative (CmRDT) : basés sur des propriétés des opérations
- Convergent (CvRDT) : basés sur des opérations de fusion d'état
- Delta (DdRDT) : un peu entre les deux

CmRDT

- On propage toutes les opérations de mise à jour à tous les réplicas
- Il peut y avoir des mise-à-jour concurrentes
- Les lectures sont faites immédiatement, sans signalement
- La diffusion peut être basée sur du multicast, gossip, ...
- mais avec les propriétés suivantes
  - fiable (tout le monde reçoit une fois exactement chaque opération)
  - causalité (chère) dans certains cas (ensembles avec retrait)

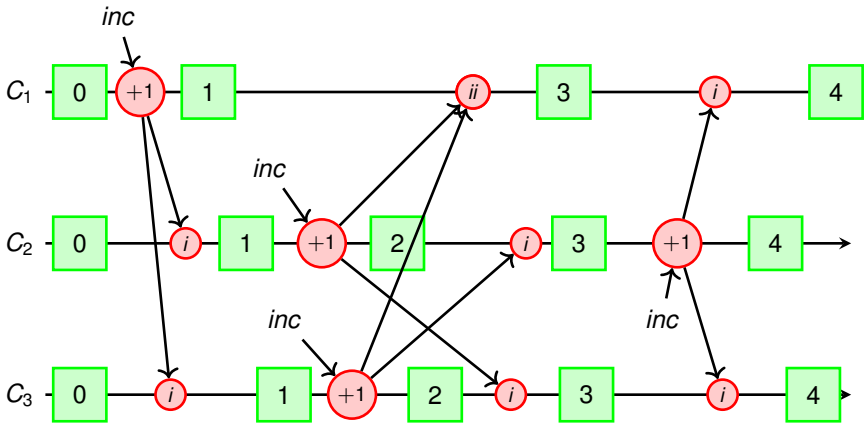
Exemple : un compteur simple

- Un compteur unique
- Initialement à 0
- Ne réalise que des incréments ($+1$)
- Ex : compteurs de *like* des réseaux sociaux

On peut compliquer avec une seconde opération de décrément (-1)

Les ensembles sont aussi des données très étudiées pour les CRDT

CmRDT d'un compteur



Analyse (de cet algorithme)

- Convergence à terme – OUI (vers 4), car toutes les opérations sont diffusés à tout le monde
- Correction – OUI
- Concurrence – OUI
- Résistance aux pertes de message – NON, il faut une diffusion fiable

CvRDT

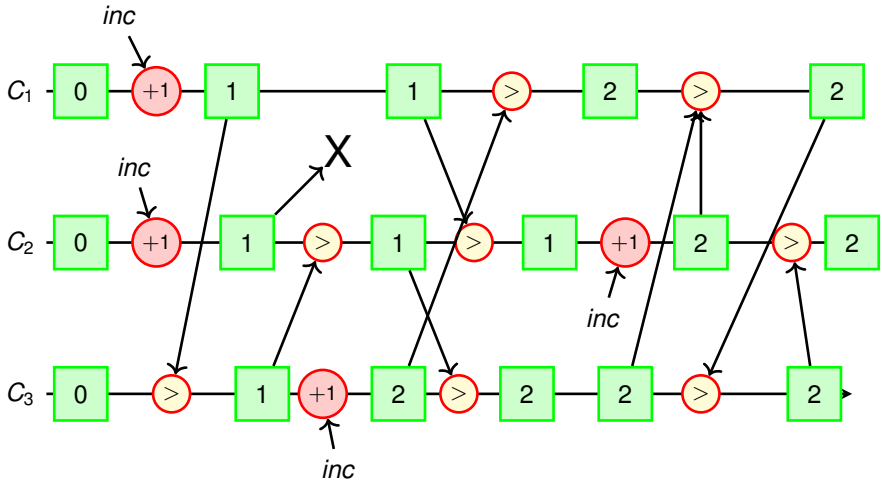
- On propage son état lors de tous les changement d'états à tous les réplicas
- On connait une opération de fusion (*merge*)
- La clé réside dans un encodage de l'état qui assure que
  - le merge est idempotent ($m = m \circ m$)
  - si deux états contiennent les effets de deux modifications différentes qui se chevauchent, le résultat du merge doit être le même que si chaque opération avait été exécutée une fois seulement (ex. add/rm dans un ensemble)

- L'encodage consomme en général de l'espace...
- La diffusion de l'état peut être variable (à chaque fois, de temps en temps)
- On peut perdre des messages ou en recevoir plusieurs grâce à l'idempotence de la fusion $\Rightarrow$ bonne résistance aux pannes

CvRDT d'un compteur

```
class Counter(CvRDT):
    ...
    def merge(self, other):
        return Counter(max(self.value(), other.value()))
```

CvRDT d'un compteur



Analyse (de cet algorithme)

- Convergence à terme – OUI (vers 2), car les réplicas transmettent leurs valeurs à tous les autres.
- Correction – NON (on devrait avoir 4), car le *max* peut masquer des incréments
- Concurrence – OUI
- Résistance aux pertes de message – OUI

Qu'est-ce qui ne va pas ?

Analogie avec une horloge de Lamport et la causalité ?

Encodons différemment... Un vecteur de compteur

CδRDT

- Entre opération et état,
- On diffuse des différences d'état (δ- state)
- Une opération de fusion sur les δ est utilisée

CRDT synthèse

- Choix de l'encodage
- Calcul local, ou accès direct à la valeur
- Stockage des connaissances locales, opération de fusion

	Op-based	State-based	δ-based
Mémoire replicas	faible	moyenne	élevée
Puissance calcul	faible	élevé	élevé
Communication	diff. fiable	R non-fiable	R non-fiable
Mémoire des buffers	faible	–	moyenne

Avancement

Conjecture CAP

Les systèmes ne peuvent avoir simultanément que 2 des 3 propriétés suivantes

- C : *consistency* – cohsérence
- A : *availability* – disponibilité
- P : *partition* – tolérance au partitionnement de réseau

Par exemple

- CA : *two phase commit*
- CP : consensus (paxos)
- AP : cohérence éventuelle