Systèmes et Algorithmes répartis

Causalité

- L'émission d'un message précède causalement sa réception : Si e = émission(m) et e' = réception(m), alors $e \prec e'$.

Coupure cohérente

Coupure cohérente

Une coupure C est cohérente si $\forall e \in C : \forall e' : e' \prec e \Rightarrow e' \in C$

- Une réception n'est pas présente sans son émission

Pour rendre une coupure cohérente, on doit ajouter un message avec une réception dans la coupure et une émission en dehors de la coupure (après).

Horloge manuelle

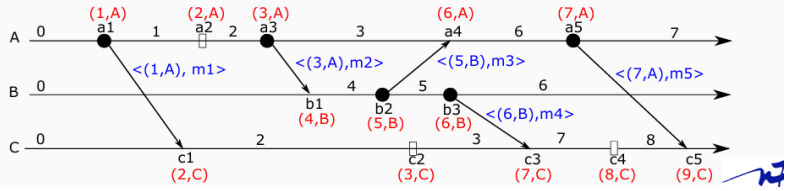
Solutions

- Synchronisation des horloges locales :
  invariant $\max_{i=1..N}(h_i) - \min_{i=1..N}(h_i) < \epsilon$
- Causalité respectée si ϵ est inférieur au temps de transmission d'un message
- Unicité en utilisant couple $(date, id\ du\ site)$

Horloge de Lamport

Chaque site s possède une horloge entière H_s. Chaque événement est daté avec le couple (H_s après l'action, id du site).

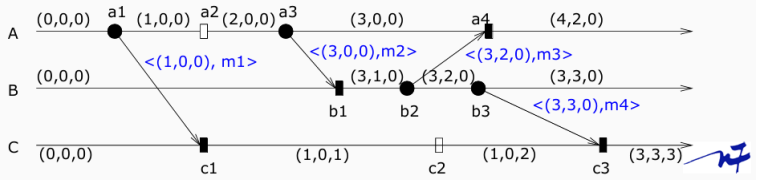
Type d'événement sur un site s	Action sur le site s
Événement interne sur s	$H_s \leftarrow H_s + 1$;
Émission sur s de m	$H_s \leftarrow H_s + 1$; envoi de $\langle\langle H_s, s\rangle, m\rangle$;
Réception sur s de $\langle\langle dm, s'\rangle, m\rangle$	$H_s \leftarrow \max(H_s, dm) + 1$;



Horloge de Lamport vectorielle

Chaque site s possède une horloge vectorielle H_s.
Chaque événement est daté avec le résultat de l'action.

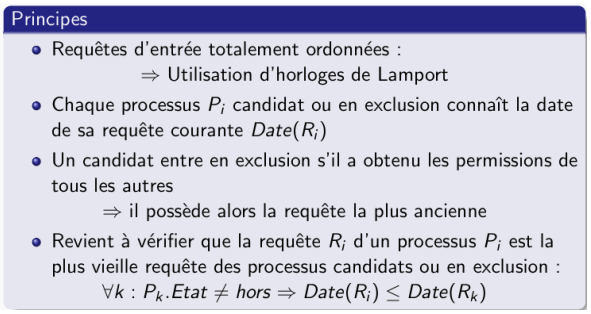
Type d'événement sur un site s	Action sur le site s
Événement interne sur s	$H_s[s] \leftarrow H_s[s] + 1$
Émission sur s de m	$H_s[s] \leftarrow H_s[s] + 1$ envoi de $\langle H_S, m\rangle$
Réception sur s de $\langle dm, m\rangle$	$H_s[s] \leftarrow H_s[s] + 1$ $H_s[s'] \leftarrow \max(H_s[s'], dm[s']), \forall s' \neq s$



Histoire causale

On prend le site de l'événement émetteur et tous les messages précédents sur le site.

L'exclusion mutuelle de Ricart-Agrawala



→ Recaler l'horloge : Cela garantit que les événements dans le système distribué respectent l'ordre causal.

→ Horloge de lamport > Horloge vectorielle :

L'algorithme a simplement besoin d'un ordre total pour décider quel processus est prioritaire (celui avec le plus petit horodatage).

Les horloges de Lamport suffisent pour cela, car elles garantissent un **ordre total** des événements en utilisant des timestamps logiques simples, alors que les horloges vectorielles ne garantissent qu'un ordre partiel.

→ Si défaillance du système, que se passe t'il ?

Interblocage car les processus en attente vont attendre indéfiniment.

→ Un site peut renvoyer son message de requête (avec la même date) s'il n'a pas reçu un message de permission au bout d'un certain temps.

Un site peut envoyer plusieurs autorisations Perm, mais une seule sera utilisée pour accéder à la section critique. Les autres, dues à des retransmissions ou des retards, pourraient rester dans le réseau et être utilisées pour satisfaire une future requête, perturbant l'ordre causal et risquant de violer l'exclusion mutuelle.

→ Solution au problème précédent

mettre la date de la requête dans le message de permission pour distinguer une retransmission d'un nouveau message.

→ Modification algorithme et ajout de détecteur de défaillance parfait (complétude forte, exactitude forte)

Un site suspect est retiré de D. Comme le détecteur est parfait, tout site défaillant finira par être enlevé et on n'enlève pas à tort un site. (maintenir la sûreté (exclusion mutuelle) et la vivacité (pas de blocage)).

→ Ajout de détecteur de défaillance ◊P (complétude forte, exactitude finalement forte)

Les précédentes modifications ne sont pas correctes : pendant un certain temps, P se trompe en suspectant à tort des processus corrects dont on n'attendra pas la permission. L'algo ne garantit que l'éventuelle exclusion mutuelle.

Généralisation N - L sites :

→ Attendre seulement N−L autorisations généralise correctement l'algorithme pour autoriser jusqu'à L sites en section critique.

Si un site reçoit N−L autorisations, cela signifie que L - 1 autres sites au maximum peuvent être en section critique simultanément.

Cette généralisation est correcte car elle garantit que :

- Au plus L sites peuvent accéder à la section critique simultanément.
- Les autorisations sont basées sur l'ordre des requêtes grâce aux horloges de Lamport.

→ Pourquoi l'algorithme est faux si un site qui libère l'accès exclusif envoie une simple autorisation à toutes les requêtes en attente

Si un site qui libère l'accès exclusif envoie une simple autorisation **sans tenir compte des horodatages**, cela peut entraîner un **désordre dans les requêtes**, et plusieurs sites pourraient accéder simultanément à la section critique de manière incohérente.

Ce problème est dû aux **autorisations tardives**, car les horodatages des requêtes ne sont pas respectés.

→ Solution au problème précédent

Maintenir une file des requêtes en attente, triée par horodatage (Lamport).

Envoyer une autorisation uniquement à la **requête la plus ancienne** dans cette file, et non à toutes les requêtes.

Consensus

Soit un ensemble de processus $p_1, \ldots, p_n$ reliés par des canaux de communication.
Initialement : chaque processus p_i propose une valeur v_i.
À la terminaison de l'algorithme : chaque p_i décide d'une valeur d_i.

- Accord : la valeur décidée est la même pour tous les processus corrects
- Intégrité : tout processus décide au plus une fois (sa décision est définitive)
- Validité : la valeur décidée est l'une des valeurs proposées
- Terminaison : tout processus correct décide au bout d'un temps fini

Consensus simultané

- Terminaison simultanée : tous les processus corrects décident *en même temps* (= au même tour, modèle synchrone).

Consensus uniforme

- Accord uniforme : la valeur décidée est la même pour tous les processus (corrects ou ultérieurement défaillants) qui décident

Synchrone

borne supérieure connue sur le temps de transmission et sur l'avancement des processus.

Asynchrone

Pas de borne connue : avancement arbitrairement lent des processus et du réseau.

Impossibilité du consensus en asynchrone avec arrêt

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

Impossibilité du consensus en synchrone avec perte de message

Le consensus est impossible à réaliser dans un système synchrone ou les messages peuvent etre arbitrairement perdus.

Détecteur de défaillance

Complétude forte : il existe un moment où tous les crashes sont suspectés → on cesse d'attendre leur message.
Exactitude forte : aucun correct n'a été suspecté à tort → on a attendu leur message.

Mémoire partagée

cohérence stricte:
Toute lecture sur une (copie d'une) donnée x renvoie une valeur correspondant à l'écriture la plus récente sur x.
cohérence séquentielle:
la lecture et l'écriture pour se font dans une séquence sans forcément respecter l'ordre.
linéarisabilité :
la lecture et l'écriture se font dans une séquence mais en respectant l'ordre

P_1 :	W(x)a					
P_2 :			W(x)b			
P_3 :		R(x)a				R(x)b
P_4 :				R(x)a	R(x)b	

linéarisable :
S =W(x)a R_3(x)a R_4(x)a W(x)b R_3(x)b R_4(x)b

P_1 :		W(x)a				
P_2 :	W(x)b					
P_3 :			R(x)a			R(x)b
P_4 :					R(x)a	R(x)b

séquentiel, mais non linéarisable (pas d'ordre série commençant par W(x)b) :
S =W(x)a R_3(x)a R_4(x)a W(x)b R_3(x)b R_4(x)b

P_1 : W(x)a						
P_2 :		W(x)b				
P_3 :			R(x)a			R(x)b
P_4 :				R(x)b	R(x)a	

non séquentiel

Cohérence causale:
→ La cohérence causale signifie que si une opération (lecture ou écriture) dépend d'une autre (par exemple, une écriture précédant une lecture), tous les processus doivent observer ces opérations dans le même ordre causal.
→ Si une écriture W(x) cause une lecture R(x), alors tous les processus doivent d'abord voir l'écriture W(x), puis la lecture R(x).
→ Une écriture doit être avant la lecture d'une lettre.
Cohérence FIFO:

Condition de cohérence FIFO

Des écritures réalisées par un même processus doivent être vues par tous les processus dans leur ordre de réalisation.

Note : les écritures réalisées par des processus différents peuvent être vues dans un ordre différent

P_1 : W(x)a								
P_2 :		R(x)a	W(x)b	W(x)c				
P_3 :					R(x)a		R(x)b	R(x)c
P_4 :						R(x)b	R(x)a	R(x)c

non causalement cohérent, mais
cohérence FIFO (R(b) précède R(c) partout ; R(a) est indépendant)