Cours ASPD
Temps et datation dans les systèmes répartis
Protocoles de groupe et protocoles d'exclusion mutuelle
Telecom Nancy 2A Apprentissage

Dominique Méry
Telecom Nancy
Université de Lorraine

Année universitaire 2024-2025

3 mars 2025

Summary

Section Courante

]

Raisonner sur un système réparti

- Définir un état du système réparti : état local, état des communications, ...
- Ordonner les différentes actions ou événements du système réparti mais maintenir la cohérence des données.
- Tenir compte des niveaux d'abstraction et des couches

Raisonner sur un système réparti

- Définir un état du système réparti : état local, état des communications, ...
- Ordonner les différentes actions ou événements du système réparti mais maintenir la cohérence des données.
- Tenir compte des niveaux d'abstraction et des couches

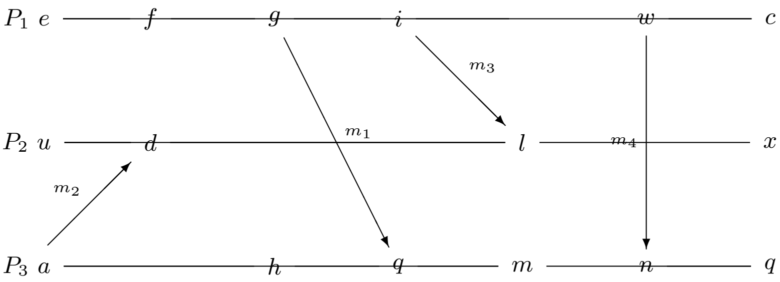
Motivation principale

Les mécanismes de communication point à point sont généralisés à des communication de groupe :

- un processus p envoie un message m à un groupe de processus D : protocole de diffusion
- propriétés attendues de ce typoe de protocole
    - validité : toute diffusion d'un message m par un processus p non-fautif à un groupe D conduit fatalement à la délivrance du message par tous les membres non-fautofs du groupe.
    - accord : si un processus non-fautif délivre le message m, alors tous les membres non-fautifs délivrent m.
    - intégrité : un message m est délivré au plus une fois à tout processus non-fautif, et seulement s'il a été diffusé par un processus.

Datation des événements

- Datation induite par l'horloge locale :
  - ordre des événements = ordre de la suite des instructions
  - e arrive avant f : e est exécuté sur le site S avant f.
  - $e \rightsquigarrow f$ signifie $e; f$
- Datation sur des sites différents : comment définir un ordre sur les événements ?



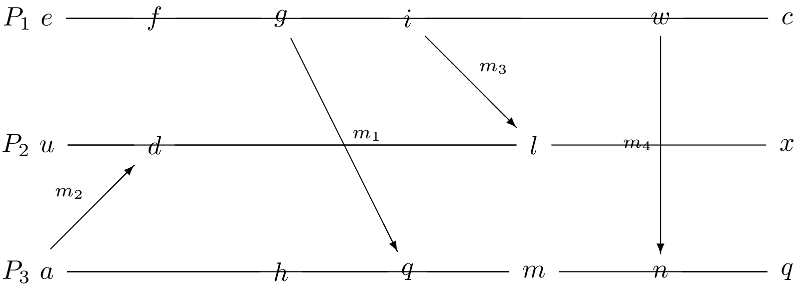
  - Des horloges locales mais pas d'horloge globale

$\rightsquigarrow$

- Construire un ordre sur les événements conservant les ordres :
  - l'ordre sur les processeurs
  - l'ordre sur les opérations d'envoi de message
- Solution de Lamport en 1978 : causalité
- $e \rightsquigarrow f$: e précède f ou e arrive avant f

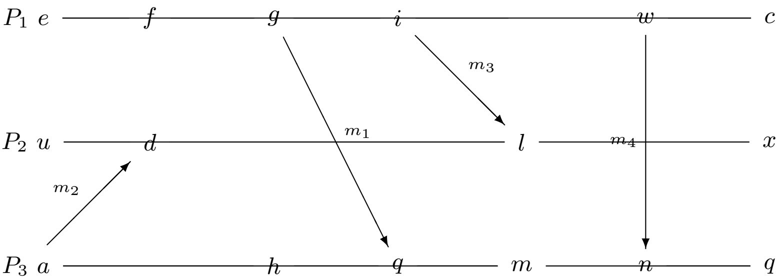
Règles de construction de la relation $\leadsto$

- **Règle 1** : Si deux événements a et b d'un même processus et si le temps d'occurence de a est antérieur à b, alors $a \leadsto b$.
- **Règle 2** : Si a est un événement d'envoi d'un message par un processus P à un processus Q et si b est l'événement de réception du même message, alors $a \leadsto b$:
- **Règle 3** : Si $a \leadsto b$ et $b \leadsto c$, alors $a \leadsto c$.

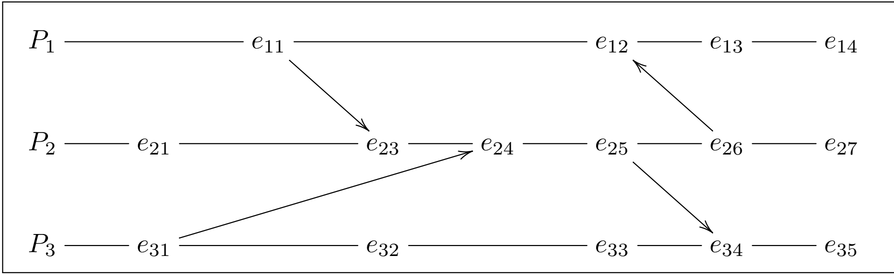


Ordre causal

- Pour chaque événement $e \in E$, on définit :
  - $Past(e) = \{e' \in E | e' \rightsquigarrow e\}$: passé de l'événement e.
  - $Future(e) = \{e' \in E | e \rightsquigarrow e'\}$: futur de l'événement e.
  - $Concurrent(e) = (E - \{e\}) - (Future(e) \cup Past(e))$: événements indépendants de e
  - $e \parallel f$ ou $e \# f$: $e' \in Concurrent(e)$ ou $e \in Concurrent(e')$
  - Propriété : $e \in Concurrent(e')$ si, et seulement si, $e \in Concurrent(e')$

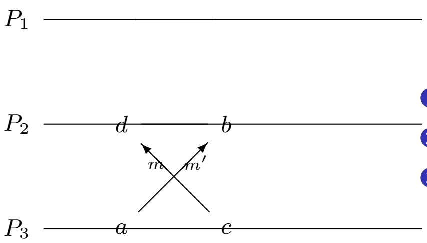


Exemple de datation



Événements d'un système

- e : événement local à un processus donné
- $Emission_i(m, j)$: émission d'un message m par le processus i.
- $Send_i(m, j)$: événement d'envoi dans le processus i d'un message m à un autre processus.
- $Receive_i(m, j)$: événement de réception par le processus i d'un message m provenant d'un autre processus.
- $Deliver_i(m, j)$: événement marquant la livraison effective du message m par le processus i.



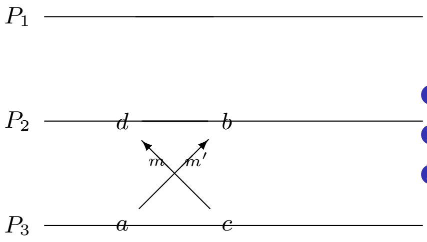
1. a envoie un message à b
2. c envoie un message à d
3. Anomalie : l'ordre d'envoi n'est pas respecté par la réception.

Réception FIFO

Réception FIFO

Si un message m est envoyé par un processus P avant un message m', alors le message m est reçu par le processus Q avant de recevoir le message m' ou encore
Si $Send_i(m,j) \leadsto Send_i(m',j)$, alors $Receive_j(m,i) \leadsto Receive_j(m',i)$

anomalie FIFO



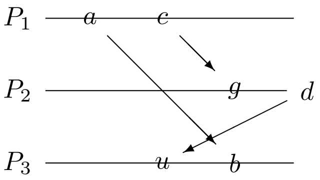
1. a envoie un message à b
2. c envoie un message à d
3. Anomalie : l'ordre d'envoi n'est pas respecté par la réception.

Livraison CAUSALE

Livraison CAUSALE

Si un message m est envoyé par un processus P avant un message m' envoyé après pour l'ordre causal, alors le message m est reçu par le processus Q avant de recevoir le message m' ou encore :
Si $Send_i(m,j) \leadsto Send_k(m',j)$, alors
$Deliver_j(m,i) \leadsto Deliver_j(m',k)$

anomalie CAUSALE

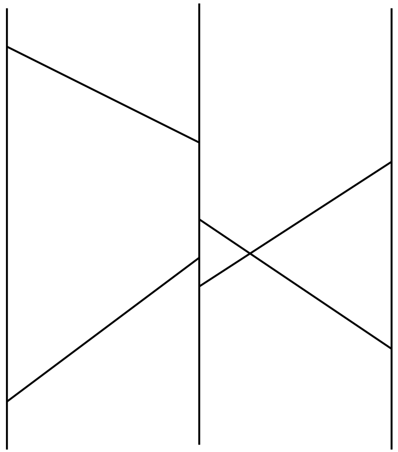


1. c envoie un message à g
2. d envoie un message à u
3. g précède d
4. Anomalie : a précède c et u précède b.

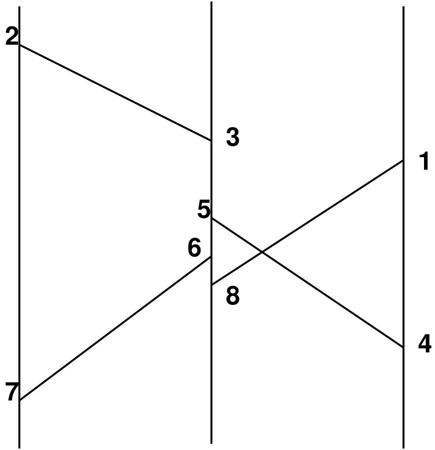
Horloge Logique

- LC désigne une horloge logique (**Logical Clock**)
- LC est une fonction associant à tout événement une valeur entière.
- Toute occurence d'un événement interne à un processus conduit à l'incrément de 1 de l'horloge locale.
- Quand on envoie un message, on ajoute la valeur de LC au message envoyé.
- Quand on reçoit un message, la valeur de LC est positionnée au maximum de l'horloge locale et de la valeur du message plus 1.
- **Propriété des horloges logiques** : Si $a \leadsto b$, alors $LC(a) < LC(b)$.

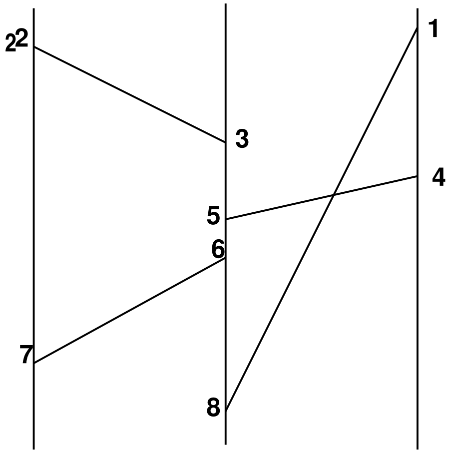
Horloge Logique : relation initiale



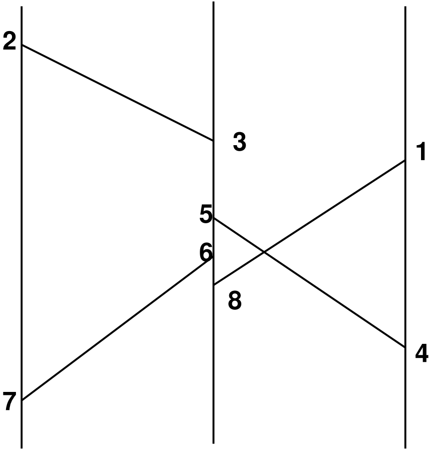
logique



Horloge Logique : réordonnancement



Horloge Logique : réordonnancement



Gestion des horloges logiques

- Définition
  - Chaque site i dispose d'une horloge locale noté LC_i : $LC_i \in E_i \nrightarrow \mathbb{N}$
  - Une fonction d'estampillage est définie et notée TS et associe à tout message une valeur : $TS \in M \nrightarrow \mathbb{N}$
  - Pour tout événement local e à un processus i : $e \in dom(LC_i)$.
  - Tout message m envoyé est estampillé par une valeur définie par une fonction d'estampillage : $m \in dom(TS)$.
  - Pour chaque événement e, on notera $P(e)$ le processus d'exécution de e.
- Opérations
  - Initialement, les horloges sont définies par 0 pour le premier événement qui existe toujours et qui correspond à l'initialisation en i : $start_i \in P(i)$ et $start(i) \in dom(LC_i)$ et $LC_i(start(i)) = 0$
  - Si un événement e est local à i, alors $LC_i(e) := Max(ran(LC_i))+1$.
  - Si un événement e est l'envoi d'un message m, alors $LC_i(e) := Max(ran(LC_i))+1$ et $TS(m) := LC_i(e)$
  - Si un événement e est la réception d'un message m, lors on met à jour l'horloge locale : $LC_i(e) := Max(TS(m), Max(ran(LC_i)))+1$.

Exemple de datation

- Chaque événement est affecté d'un entier

Exemple de datation

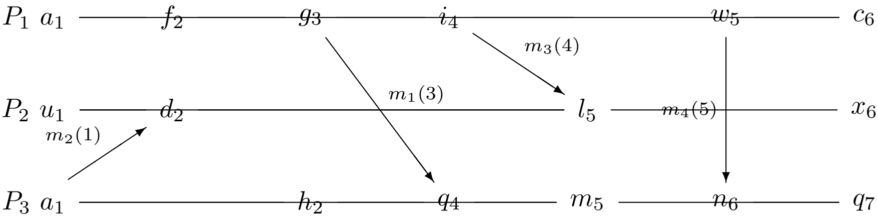
- Chaque événement est affecté d'un entier
- Deux événements peuvent avoir la même valeur

Exemple de datation

- Chaque événement est affecté d'un entier
- Deux événements peuvent avoir la même valeur

Exemple de datation

- Chaque événement est affecté d'un entier
- Deux événements peuvent avoir la même valeur



Définition d'un ordre strict

- L'estampille d'un événement e est définie par la paire $(LC_{P(e)}(e), P(e))$.
- On définit un ordre strict sur les estampilles par $LC(e) = (LC_{P(e)}(e), P(e))$.
  - $LC(e) \prec LC(f)$ ou $(LC_{P(e)}(e), P(e)) \prec (LC_{P(f)}(f), P(f))$:
    $$\left\{ \begin{array}{ll} & LC_{P(e)}(e) < LC_{P(f)}(f) \\ \textbf{ou} & (LC_{P(e)}(e) = LC_{P(f)}(f)) \textbf{ et } (P(e) < P(f) \end{array} \right.$$
- $\prec$ est transitive
- **Propriété** : Si $e \leadsto f$, alors $LC(e) \prec LC(f)$.

Démonstration de la propriété

Propriété

Si $e \leadsto f$, alors $LC(e) \prec LC(f)$.

Démonstration Récurrence sur la longueur de la suite $e \leadsto f$.

- $e \leadsto f$ est de longueur 1 : deux cas sont envisagés
  - e et f sont deux événements locaux et se suivent : $LC(f) = LC(e)+1$ □
  - e est un envoi de m et f est la réception de m : $LC(f) = Max(TS(m), Max(ran(LC_i)))+1$ et $TS(m) = LC_{P(e)}(e)$. On en déduit que $LC(e) \prec LC(f)$. □
- $e \leadsto f$ est de longueur strictement plus grande que 1 : Dans ce cas, on a une suite de longueur n telle que $e = e_0 \ldots e_i \ldots e_n = f$ d'événements liés par la relation $\leadsto (\forall i \in \{0 \ldots n-1\}.e_i \prec e_{i+1}$.
  - Par hypothèse de récurrence et par construction de $\leadsto$, on déduit que $e \leadsto e_{n-1}$ et $LC(e) \prec LC(e_{n-1})$.
  - Puisque $e_{n-1} \prec e_n$, on analyse les deux cas comme pour le pas de récurrence initiale pour établir que $LC(e_{n-1}) \prec LC(e_n)$
  - Par transitivit de la relation $\prec$, on établit que $LC(e) \prec LC(f)$. □

Section Courante

]

estampillage

- e est local à i : $l(e) \stackrel{def}{=} e \wedge clock' = clock[i|-> clock[i]+1]$
- e est un envoi de i à j : $l(e) \stackrel{def}{=} clock' = clock[i|-> clock[i]+1] \wedge e \wedge file' = file[i,j|-> < m, clock[i]+1 >]$
- e est une réception par i de j : $l(e) \stackrel{def}{=} < m, c >= file[i,j] \wedge e \wedge file' = file[i,j|-> <>] \wedge clock' = clock[i|-> Max(clock[i], c > +1]$

Estampillage TLA$^+$

- Une estampille (timestample) est une couple formé d'une valeur entière positive et d'un entier naturel.
- Les estampilles sont comparables par la relation d'ordre totale suivante :
  $< c, i > << < d, j > > \stackrel{def}{=} (c < d) \vee (c = d \wedge i < j)$
- Si un système réparti satisfait une propriété d'invariance I, alors le système transformé satisfait I.
- Si un système réparti satisfait une propriété de sûreté S, alors le système transformé satisfait S.
- Tout système transformé est un raffinement du système transformé.

Section Courante

]

Vecteur d'horloges

- **Propriété des horloges logiques** : Si $a \leadsto b$, alors $LC(a) \prec LC(b)$.
- L'ordre $\prec$ ne permet pas, en comparant les valeurs des horloges, de déduire une causalité entre 2 événements.
- Pour y remédier on va associer aux événements un vecteur d'horloges qui permettra de décider, s'il y a une relation de causalité entre 2 événements :

Objectif des horloges vectorielles

$a \leadsto b$ si, et seulement si, $VC(a) \prec_v VC(b)$.

e et g ne sont pas comparables pour $\prec$

Example

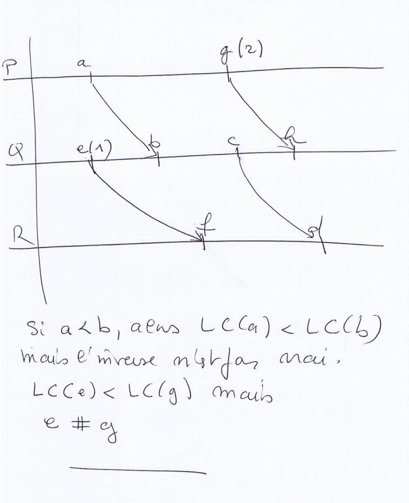


Illustration des vecteurs d'horloge

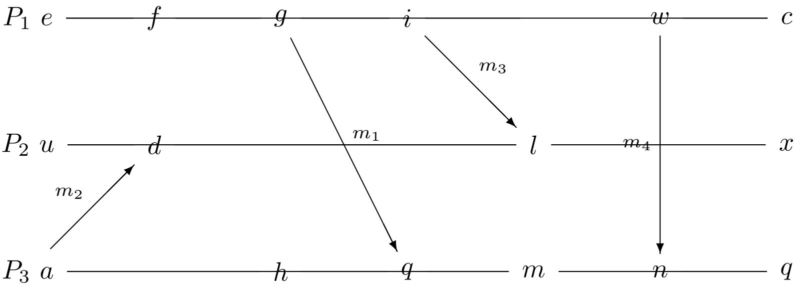


Illustration des vecteurs d'horloge

Illustration des vecteurs d'horloge

devient décoré comme suit

Illustration des vecteurs d'horloge

devient décoré comme suit

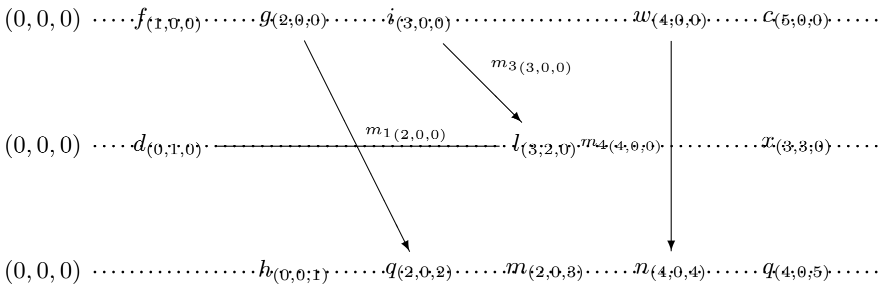
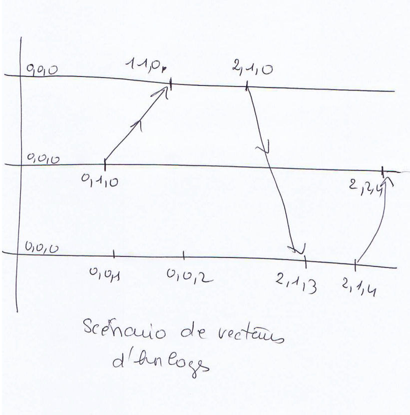


Illustration 2



Vecteur d'horloges (I)

Contexte

- Chaque site $i \in \{1 \ldots n\}$ dispose d'un vecteur local noté VC_i : $VC_i \in E_i \pto \mathbb{N}^n$
- Une fonction d'estampillage est définie et notée MC et associe à tout message une valeur : $MC \in M \pto \mathbb{N}^n$
- Pour tout événement local e à un processus i : $e \in dom(VC_i)$.
- Tout message m envoyé est estampillé par une valeur définie par une fonction d'estampillage : $m \in dom(MC)$.
- Pour chaque événement e, on notera $P(e)$ le processus d'exécution de e.

Opérations sur les n-uplets d'entiers

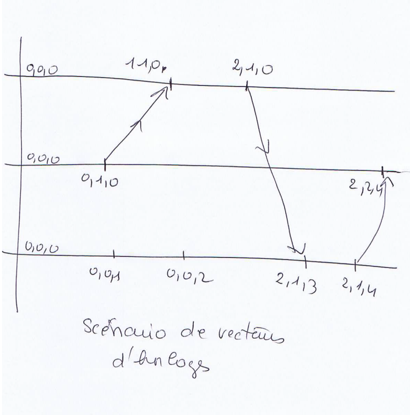
- $v \in 1..n \rightarrow \mathbb{N}$ et $w \in 1..n \rightarrow \mathbb{N}$
- $1_i \in 1..n \rightarrow \mathbb{N}$:
  - $1_i(i) = 1$
  - $\forall j.j \in \{1..n\} \wedge j \neq i \Rightarrow 1_i(j) = 0$
- $v \oplus 1_i \in 1..n \rightarrow \mathbb{N}$:
  - $(v \oplus 1_i)(i) = v(i)+1$
  - $\forall j.j \in \{1..n\} \wedge j \neq i \Rightarrow (v \oplus 1_i)(j) = v(j)$
- $v \leq_{uple} w : \forall j.j \in \{1..n\} \Rightarrow v(j) \leq w(j)$
- $Max \in POW(1..n \rightarrow \mathbb{N}) \nrightarrow 1..n \rightarrow \mathbb{N}$:
  - $dom(Max) \subseteq POW(1..n \rightarrow \mathbb{N}) - \{\varnothing\}$
  - Max renvoie la valeur maximale (selon l'ordre $\leq_{uple}$) d'un ensemble fini de n-uplets, si elle existe.
  - $Max(\{(0,1,0),(3,4,0),(7,0,9)\})$ n'existe pas.
  - $Max(\{(0,1,0),(3,4,0),(7,6,9)\}) = (7,6,9)$.
- Pour tout n-uple u ($\in 1..n \rightarrow \mathbb{N}$), on définit la j-ième projection par la notation $u(j)$.
  - $(0,6,5)(2) = 6$, $(0,6,5)(1) = 0$, $(0,6,5)(3) = 5$

Vecteur d'horloges (II)

Mécanisme

- Initialement, les horloges sont définies par $(0,0,0)$ pour le premier événement qui existe toujours et qui correspond à l'initialisation en i : $start_i \in P(i)$ et $start(i) \in dom(VC_i)$ et $VC_i(start(i)) = (0,0,0)$
- Si un événement e est local à i, alors $VC_i(e) := Max(ran(VC_i)) \oplus 1_i$.
- Si un événement e est l'envoi d'un message m, alors $VC_i(e) := Max(ran(VC_i)) \oplus 1_{P(e)}$ et $MC(m) := VC_i(e)$.
- Si un événement e est la réception d'un message m, alors
  1. on met à jour le vecteur local : $VC_i(e)(i) := Max(ran(VC_i), MC(m)) + 1$
  2. et $\forall j. j \in 1..n \land j \neq i \Rightarrow VC_i(e)(j) := Max(Max(ran(VC_i)), MC(m))(j)$.

Illustration 2



Propriétés de la datation vectorielle

Date d'un événement e et sens de $VC_{P(e)}(e)$

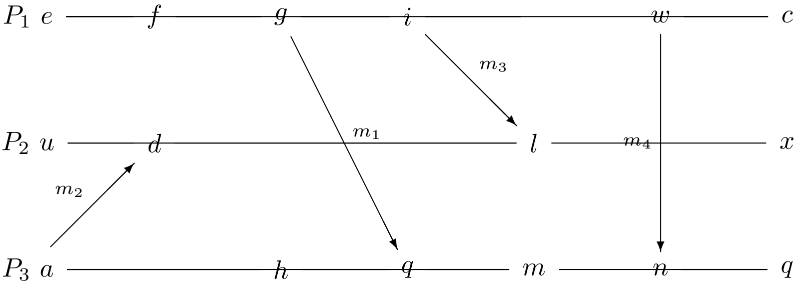
La valeur de la ième composante de $VC_{P(e)}(e)$ correspond au nombre d'événements dans la passé de e pour le site i ou que e connaît.

Propriétés

Soit un événement $e \in E$ tel que $e \in dom(VC_{P(e)})$ (signifiant que e a une date ou encore qu'il a eu une occurence).

- Si $i \neq P(e)$, alors $VC_{P(e)}(e)(i) = Card(\{e' | e' \in E_i \wedge e' \leadsto e\})$
- Si $i = P(e)$, alors $VC_{P(e)}(e)(i) = Card(\{e' | e' \in E_i \wedge e' \leadsto e\})$

Illustration des vecteurs d'horloge



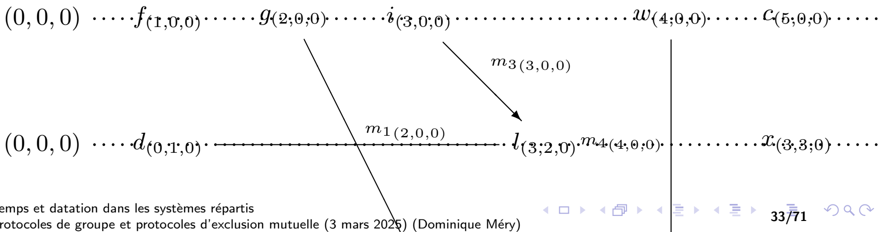
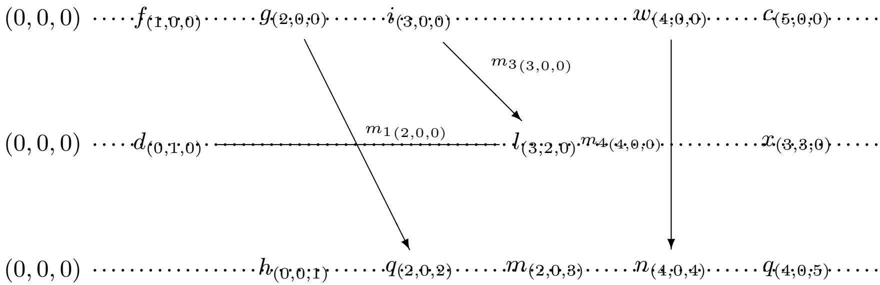


Illustration des vecteurs d'horloge



Propriétés

I

Soient e_1 et e_2 2 événements se produisant dans le réseau.
Pour tous les événements e et f de E,

$$e \leadsto f \text{ si, et seulement si, } VC_{P(e)}(e) \leq_{uple} VC_{P(f)}(f)$$

II

Soient e_1 et e_2 2 événements se produisant dans le réseau.
Pour tous les événements e et f de E,

$$e\#f \text{ si, et seulement si, } VC_{P(e)}(e) \text{ et } VC_{P(f)}(f) \text{ ne sont pas comparables par } \leq_{uple}.$$

Justification (I)

$e \leadsto f$: $Past(e) \subseteq Past(f)$ et $\forall i.i \in 1..n \Rightarrow VC_{P(e)}(i) \leq VC_{P(f)}(i)$

$VC_{P(f)}(f)$:

Justification (II)

$e\#f$

par $\leq_{uple}$.

Propriété des horloges vectorielles

densité

Soient deux événements e_i de P_i et e_j de P_j.
Si $VC(e_i)(k) < VC(e_j)(k)$, alors il existe un événement e tel que $\neg(e \longrightarrow e_i)$ et $e \longrightarrow e_j$.

Il existe un événement e qui a permis l'incrément de la composante k de l'horloge verctorielle. e n'est pas la cause de e_i.

Section Courante

]

Etat courant

Problèmes d'exclusion mutuelle

PROBLÈME : un ensemble d'agents communicants souhaitent partager une ressource commune et

- avoir accès au bout d'un temps fini après une requête
- avoir un accès exclusif
- être considéré de manière équitable

Problème de l'Exclusion mutuelle

- Assurer le partage de ressources communes
- Garantir une répartition équitable de ces ressources partagées
- Environnement centralisé solutions logicielles : algorithmes de Dekker, de Dijkstra, de Peterson,
- Environnement centralisé solutions matérielles : sémaphores, test and sets ...

Etat courant

Solutions en système centralisé

- Utilisation de verrous : lock, unlock
- Utilisation de sémaphores :
- Variables de priorités Bakery

Sémaphores

- Un sémaphore est une structure constituée d'une variable s et d'une file d'attente q et cette structure est gérée par dexu opérations :
  - $P(s)$: demande du sémaphore
  - $V(s)$: libération du sémaphore
- Propriété 1 : le nombre de processus distincts utilisant le sémaphore est d'au plus sa valeur initiale.
- Propriété 2 : tout processus demandant le sémaphore poura l'obtenir à condition qu'au moins un des processus le possédant le rende.

CONTEXT $sem0$

SETS

$PROCESSES$

CONSTANTS

$smax$

AXIOMS

$axm1 : PROCESSES \neq \varnothing$

$axm2 : smax \in \mathbb{N}$

$axm3 : smax \neq 0$

END

MACHINE $sem1$

SEES $sem0$

VARIABLES

s, f, r, get

INVARIANTS

$inv1 : s \in \mathbb{N}$

$inv2 : r \in \mathbb{N}$

$inv3 : f \in 1 \,..\, r \rightarrowtail PROCESSES$

$inv4 : get \subseteq PROCESSES$

$inv5 : s \leq smax$

$inv6 : ran(f) \cap get = \varnothing$

$inv7 : dom(f) = 1 \,..\, r$

$inv8 : finite(get)$

$inv9 : s + card(get) = smax$

$inv10 : r \neq 0 \Rightarrow s = 0$

$inv11 : s \neq 0 \Rightarrow r = 0$

$inv12 : s \neq 0 \Rightarrow dom(f) = \varnothing$

EVENT INITIALISATION

BEGIN

$act1 : s := smax$

$act2 : r := 0$

$act3 : get := \varnothing$

$act4 : f := \varnothing$

END

EVENT RequestSemFree

ANY

p

WHERE

$grd1 : p \in PROCESSES$

$grd2 : p \notin get$

$grd3 : s \neq 0$

THEN

$act1 s := s-1$

$act2 get := get \cup \{p\}$

END

EVENT RequestSemWaiting

ANY

p

WHERE

$grd1 : p \in PROCESSES$

$grd2 : s = 0$

$grd3 : p \notin get$

THEN

$act1 : f(r+1) := p$

$act2 : r := r+1$

END

EVENT ReleaseSemFree

ANY

p

WHERE

$grd1 : p \in PROCESSES$

$grd2 : p \in get$

$grd3 : r = 0$

THEN

$act1 : get := get \setminus \{p\}$

$act2 : s := s+1$

END

EVENT ReleaseSemWaiting

ANY

p, q

WHERE

$grd1 : p \in get$

$grd2 : q \in ran(f)$

$grd3 : q = f(1)$

THEN

$act1 : get := (get \setminus \{p\}) \cup \{q\}$

$act2 : r := r{-}1$

$act3 : f :|(f' \in 1 \,..\, (r{-}1) \rightarrowtail PROCESSES \land (\forall i \cdot i \in 1 \,..\, (r{-}1) \Rightarrow f'(i)$

END

Algorithmes classiques d'exclusion mutuelle

- Garantir l'exclusion mutuelle par des variables de priorités
- Plusieurs solutions existent comme Dekker, Dijkstra, ...
- Algorithme *bakery* de Lamport : rôle des variables y1 et y2.

Etat courant

Problèmes posés par la répartition

- Hypothèse 1 : le réseau est supposés complet ou encore que les sites communiquent entre eux via un protocole fiable.
- Hypothèse 2 : chaque site a sa propre horloge.
- Problème : Les sites partagent une ressource commune et la demande de ressource conduit à un service exclusif, effectif et équitable.

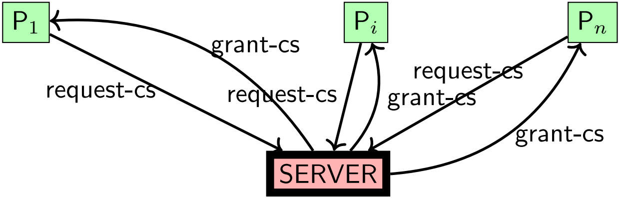
Problèmes posés par la répartition

- Hypothèse 1 : le réseau est supposés complet ou encore que les sites communiquent entre eux via un protocole fiable.
- Hypothèse 2 : chaque site a sa propre horloge.
- Problème : Les sites partagent une ressource commune et la demande de ressource conduit à un service exclusif, effectif et équitable.

Idées de solution

- Assurer l'exclusion mutuelle à l'aide d'une file d'attente qui gère les requêtes.
- Deux solutions pour la file d'attente
  - Un site joue le rôle de serveur de la file d'attente
  - La file est implicite au niveau du protocole et est gérée par tous les processus ;

Client Serveur



- SERVER gère une file d'attente et sert les demandes selon cette file d'attente.
- Les communications sont fiables ou pas

Décomposition en phases

- Phase de demande
- Phase d'attente
- Phase d'utilisation
- Phase de remise
- Phase de fin

- Chaque phase est propre à un processus séquentiel
- Chaque phase du processus P est concurente au process Q où P et Q sont distincts

Questions liées à la répartition

- Un ensemble de processus $Proc$ partagent une ressource commune R
- Tout processus est connecté à tout processus autre du réseau
- La question est de trouver un moyen d'ordonner totalement les requêtes de section critique sont totalement ordonnables.
- Trouver un ordre total : les estampilles de Lamport

Principes des algorithmes d'exclusion mutuelle

Les algorithmes d'exclusion mutuelle fonctionnent sur le modèle suivant :

- Phase de demande
- Phase d'attente
- Phase d'utilisation
- Phase de remise
- Phase de fin

Etat courant

Principes de l'algorithme d'exclusion mutuelle

- Phase de requête : p demande la section critique et envoie à tous les autres sites son estampille : $n-1$ messages sont envoyés
- Phase d'attente : p attend de recevoir un message de chaque site lui permettant d'entrer en section critique : $n-1$ messages sont reçus
- Phase de section critique : p utilise la section critique et il la rendra au bout d'un temps fini.
- Phase de relâche : le processus p sort de section critique et renvoie un message à tous les sites : $n-1$ messages sont envoyés
- Total des message : $3 \cdot n-3$ messages sont nécessaires.

Mise en œuvre

- Problème à résoudre : trouver un mécanisme équitable pour ordonnancer les demandes.
- Solution : mettre en œuvre une file d'attente de manière répartie qui permette de positionner les sites demandeurs les uns par rapport aux autres.
- Mécanisme des estampilles :
  - chaque site a un numéro propre
  - chaque site maintient un numéro de demande qui est mis à jour en fonction des demandes
  - $(p, n) < (q, m)$ si et seulement si $p < q \vee p = q \wedge n < m$

Algorithme de Lamport

- Phase de requête : p demande la section critique et envoie à tous les autres sites son estampille (diffusion aux n-1 sites).
- Phase d'attente : p attend de recevoir un message de chaque site lui permettant d'entrer en section critique (attente de n-1 réponses de site).
- Phase de section critique.
- Phase de relâche : le processus p sort de section critique et renvoie un message à tous les sites pour préciser qu'il sort et il donne une information sur son estampille aux autres processus. (diffusion aux n -1 sites).
- $3 \cdot (n-1)$ messages sont nécessaires

Algorithme de Ricart et Agrawala

- Phase de requête : p demande la section critique et envoie à tous les autres sites son estampille, .
- Phase d'attente : p attend de recevoir un message de chaque site lui permettant d'entrer en section critique.
- Phase de section critique.
- Phase de relâche : le processus p sort de section critique et renvoie un message à tous les sites qui sont en attente de son accord ; cela signifie que l'autorisation du processus p est attendue par certains sites.
  qui ont été différés pour préciser qu'il sort et il donne une information sur son estampille aux autres processus.
- $2 \cdot n - 2$ messages sont nécessaires.

Algorithme de Carvalho et Roucairol

- Même idée que Ricart et Agrawala mais avec une amélioration : un site ne demande pas aux sites qui ne lui ont pas demandé l'autorisation, sauf la première fois.
- Phase de requête : p demande la section critique et envoie à tous les autres sites son estampille mais la seconde fois n'envoie pas aux sites qui ne lui ont pas demandé.
- Phase d'attente : p attend de recevoir un message de chaque site lui permettant d'entrer en section critique.
- Phase de section critique.
- Phase de relâche : le processus p sort de section critique et renvoie un message à tous les sites qui sont en attente de son accord ; cela signifie que l'autorisation du processus p est attendue par certains sites.

  qui ont été différés pour préciser qu'il sort et il donne une information sur son estampille aux autres processus.
- $2 \cdot n - 2$ messages sont nécessaires mais au plus.

Autres solutions

- A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems MAMORU MAEKAWA University of Tokyo

Autres solutions

- A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems MAMORU MAEKAWA University of Tokyo
- Partition de l'espace des sites
- Chaque élément de la partition gère les sites de sa classe
- Demande aux représentants de classe

Conclusion et Questions

- Mécanisme de priorité
- Gestion d'une file d'attente répartie
- Mécanisme d'estampillage
- Hypothèses fortes :
  - graphe complet
  - communications fiables

Section Courante

]

Diffusion fiable

Une diffusion est fiable, si

- elle est valide : quand un processus diffuse, tous les processus membres du groupe de diffusion reçoivent.
- elle satisfait la propriété d'accord : si un processus reçoit, alors tous les autres membres du groupe reçoivent.
- elle est intègre : chaque message n'arrive qu'une et une seule fois.

Les mécanismes de diffusion fiable sont de type

- **FIFO** : les messages sont délivrés selon l'ordre d'envoi (`FBCAST`)
- **causalité** : les messages sont délivrés selon un ordre respectant la causalité (`CBCAST`)
- **Atomique** : les messages sont tous délivrés dans le même ordre (`ABCAST`)

Conventions et notations

- $\mathbf{receive}_P(m)$: événement de réception d'un message m par le processus P.
- $\mathbf{delivery}_P(m)$: événement de délivrance du message m au processus P.

Conventions et notations

- $\mathbf{receive}_P(m)$: événement de réception d'un message m par le processus P.
- $\mathbf{delivery}_P(m)$: événement de délivrance du message m au processus P.
- Observation : $\mathbf{receive}_P(m)$ précède $\mathbf{delivery}_P(m)$
- Idée : différer la délivrance d'un message quand il est reçu.

Ordre FIFO avec FBCAST

Principe

Si un processus diffuse un message m1 puis un message m2, alors aucun processus du groupe ne livre le message m2 à moins que m1 ait été livré.

- Si $\mathbf{send}_P(m1) \leadsto \mathbf{send}_P(m2)$, alors pour tout processus Q du groupe de diffusion D, $\mathbf{delivery}_Q(m1) \leadsto \mathbf{cdlivery}_Q(m2)$.
- Idée de l'algorithme : à la réception des messages, on les stocke et on compare les estampilles des messages pour la composante d'envoi P.

Ordre causal avec CBCAST

Principe

Si

- un processus envoie un message m1
- la délivrance de m1 est suivi causalement de l'envoi de m2

alors tous les processus délivrent le message m2 après le message m1.

- Si $\mathbf{delivery}_Q(m1) \rightsquigarrow \mathbf{send}_P(m2)$, alors pour tout processus Q du groupe de diffusion D, $\mathbf{delivery}_Q(m1) \rightsquigarrow \mathbf{cdlivery}_Q(m2)$.
- Idée de l'algorithme : à la réception des messages, on les stocke et on compare les estampilles des messages pour la composante d'envoi P.

Ordre atomique avec ABCAST

Principe

Les processus d'un groupe livre les messages dans le même ordre.

- Si $\mathbf{delivery}_P(m1) \leadsto \mathbf{send}_P(m2)$, alors pour tout processus Q du groupe de diffusion D, $\mathbf{delivery}_Q(m1) \leadsto \mathbf{cdlivery}_Q(m2)$.
- Idée de l'algorithme : à la réception des messages, on les stocke et on compare les estampilles des messages pour la composante d'envoi P.

Protocole de diffusion CBCAST

Initialisation

Pour chaque site $i \in 1..n$, positionner les valeurs des vecteurs VC_i à 0.

Diffusion de m sur le site i

$$\left\{\begin{array}{l} VC_i(i) := VC_i(i)+1 \\ \textbf{Pour tout site j de 1..n} : Send_i(m, VC_i(i), j) \end{array}\right.$$

Réception

$$\left\{\begin{array}{l} Receive_i(m, VC_m, j) \\ Wait(VC_m = VC_i(j)+1) \\ Wait(\forall j.j \in 1..n \wedge j \neq VC_m \leq VC_i(j)) \\ Deliver_i(m) \\ VC_i(j) = VC_i(j)+1 \end{array}\right.$$

Conclusion

- Rôle des estampilles pour les algorithmes d'exclusion mutuelle
- Rôle de horloges vectorielles dans la diffusion des messages et la propriété de causalité