Ordre, temps et état dans un système réparti (2)

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr~krakowia

Remarque (et rappel)

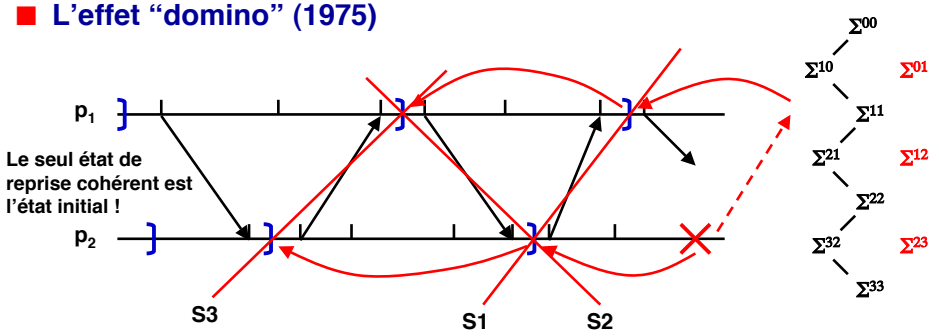
- **Conditions d'application de la présente étude**
  - Le système est asynchrone
  - Le nombre de processus est fixe et connu (ou varie de façon lente et prévisible)
    - L'opération "introduire un processus" ou "faire sortir un processus" est complexe, mais c'est acceptable si elle est peu fréquente
- **Si on n'est pas dans ce cadre, d'autres techniques sont utilisées**
  - On les verra plus tard dans ce cours

Enregistrer l'état d'un système

- **Motivations**
  - Observation, mise au point
  - Détection de propriétés (prédicats sur l'état)
  - Reprise en cas de panne ultérieure
- **Contraintes**
  - L'état enregistré doit être cohérent
  - Le coût de l'enregistrement doit être contrôlé
- **Difficultés**
  - L'enregistrement est un ensemble d'opérations locales à chaque processus ou site
  - La cohérence est une propriété globale

Un peu d'histoire

- **L'effet "domino" (1975)**



- Origine : mauvaise compréhension (à l'époque) des problèmes de cohérence d'état

Enregistrer un état cohérent

- **Deux voies d'approche**
  - "Photographier" un état : enregistrement coordonné
    - Les enregistrements locaux doivent être synchronisés pour que l'état résultant soit cohérent
    - La synchronisation a un coût à l'exécution
  - Enregistrer sans coordination et reconstruire un état cohérent à partir des enregistrements
    - Le traitement est réalisé *off line* (surcoût réduit à l'exécution)
    - Le risque d'effet domino n'est pas éliminé

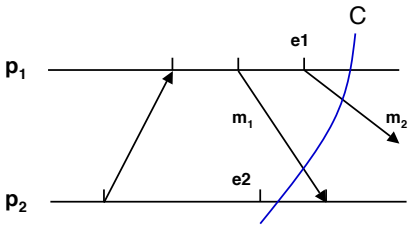
Un algorithme d'enregistrement coordonné

- **Propriétés** (Chandy-Lamport, 1985)
  - La décision d'enregistrer est prise localement par un des processus
    - Plusieurs processus peuvent lancer "simultanément" un enregistrement
  - L'algorithme se termine en temps fini
  - L'algorithme enregistre un état cohérent
    - état des processus
    - état des canaux de communication
- **Hypothèse**
  - Les canaux de communication entre processus sont FIFO
    - une variante de l'algorithme permet de lever cette restriction

État enregistré

État enregistré pour C

- état de p_1 : e1
- état de p_2 : e2
- état du canal $p_1 \rightarrow p_2$: $\{m_1, m_2\}$
- état du canal $p_2 \rightarrow p_1$: $\varnothing$

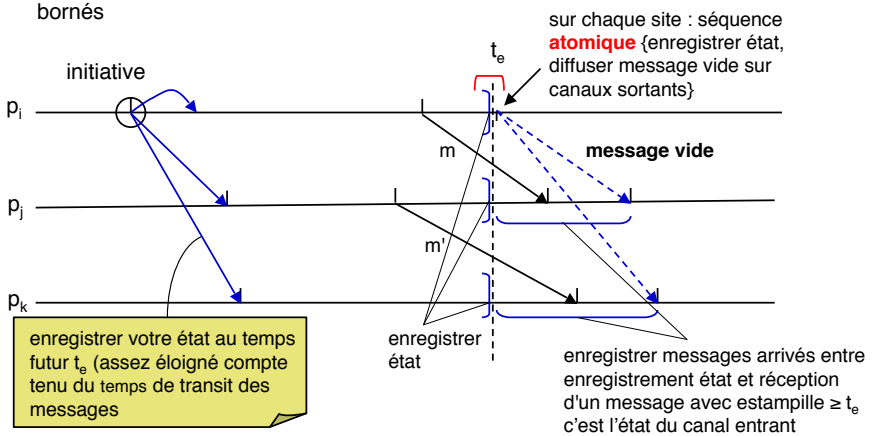


En fait, l'état des canaux peut se déduire de l'analyse de l'état des processus, mais il est plus commode de le définir explicitement

Pour construire l'algorithme de Chandy-Lamport, on va construire une suite de protocoles avec des hypothèses de moins en moins restrictives, et en préservant des propriétés entre un protocole et le suivant

Enregistrement : Protocole 1 (1)

Hypothèses
- on dispose d'une horloge HR donnant le temps réel sur chaque site
- le système est synchrone : délai de propagation et rapport des vitesses sont bornés

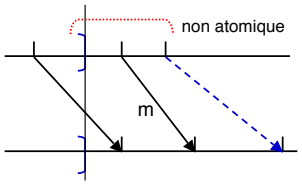


Enregistrement : Protocole 1 (2)

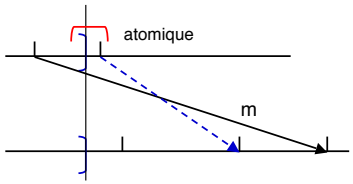
Propriétés

La coupure C_e définie par le temps t_e définit bien un état cohérent
Si e ∈ C_e et e' → e, alors HR(e') < HR(e) car HR vérifie la condition faible de validité. Donc e' ∈ C_e

L'atomicité de la séquence {enregistrement état, envoi message vide} et la propriété FIFO garantissent que l'état du canal est correct



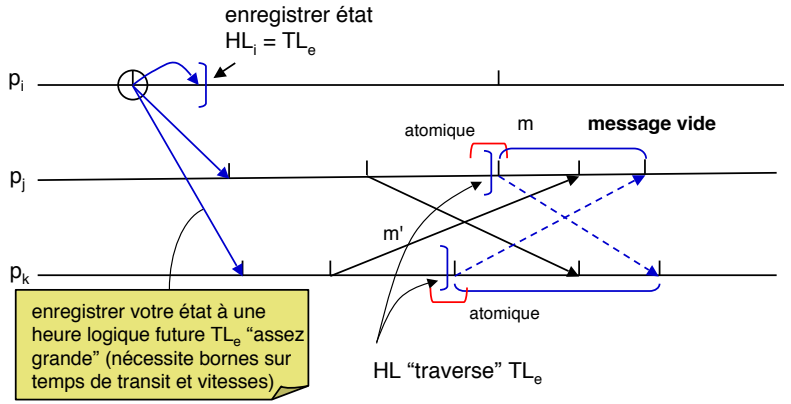
séquence non atomique : m enregistré à tort dans état canal p1→p2. (m aurait dû partir avant l'enregistrement ou après le message vide)



messages non FIFO : m non enregistré dans état canal p1→p2

Enregistrement : Protocole 2 (1)

Hypothèses
- on remplace l'horloge HR par un système d'horloges logiques HL
- le système est toujours synchrone : délai de propagation et rapport des vitesses sont bornés



Enregistrement : Protocole 2 (2)

Comment vérifier la condition : HL "traverse" TL_e :

Noter que l'incrémentation de HL est provoquée par un événement local.
- Si cet évenement est une transition d'état ou une émission, vérifier la valeur préalable de HL
- Si cet évenement est la réception d'un message, vérifier la valeur de l'estampille du message

Validité du protocole 2

La propriété de HR qui garantit que l'état enregistré est cohérent dans le protocole 1 est la condition de validité faible.

Cette condition est vérifiée par les horloges logiques

Enregistrement : Protocole 3

Dans le protocole 2 :

Pour un processus p (non initiateur), entre la réception du message "enregistrer état" et l'enregistrement lui-même, aucune action liée au protocole n'est exécutée

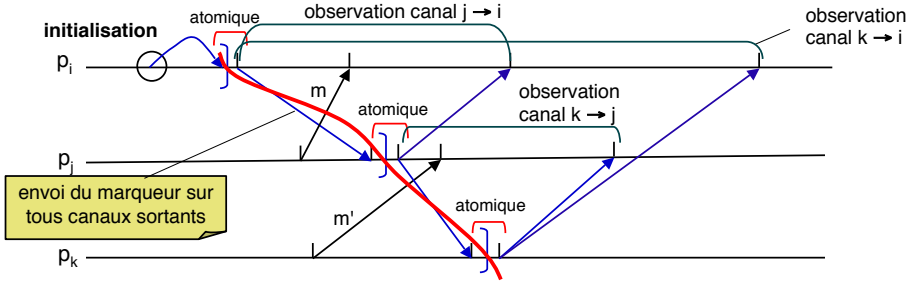
Donc on peut supprimer le délai entre les deux événements.

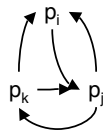
Alors TL_e devient inutile !

On peut donc aussi lever l'hypothèse de synchronisme, qui n'est utilisée que pour déterminer TL_e

En résumé, on a levé toutes les hypothèses restrictives du protocole 1

Protocole de Chandy-Lamport (1)





1-ère réception du marqueur sur p_n (depuis p_m) :
- **enregistrer état p_n**
- **envoyer marqueur sur canaux sortants** } **atomique**
- **état canal m →n = ∅**
- **commencer à enregistrer messages sur autres canaux entrants**

réception ultérieure (depuis p_q)
- **état canal q → n = messages reçus depuis enregistrement état**
- **arrêter enregistrement messages de p_q**

Protocole de Chandy-Lamport (2)

Propriétés

- **Le protocole est vivace (progrès garanti, pas de blocage)**
- **Le protocole se termine (mais sans délai garanti en asynchrone)**
- **Le protocole enregistre un état cohérent**

Quel est l'état enregistré ?

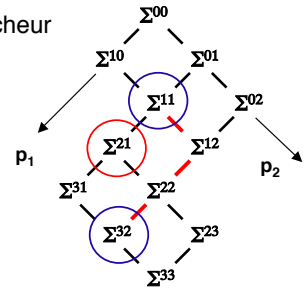
Soit S_{init} un état cohérent contenant l'événement déclencheur
Soit S_{final} un état cohérent dans lequel le protocole est terminé
Alors

$S_{init} \rightsquigarrow S_{enreg} \rightsquigarrow S_{final}$

Exemple

Si $S_{init} = \Sigma^{11}$, $S_{final} = \Sigma^{32}$, alors S_{enreg} peut être : Σ^{21}, Σ^{12}, Σ^{31}, Σ^{22}

Il est possible que l'état S_{enreg} n'ait jamais été traversé



Utilisation de l'enregistrement

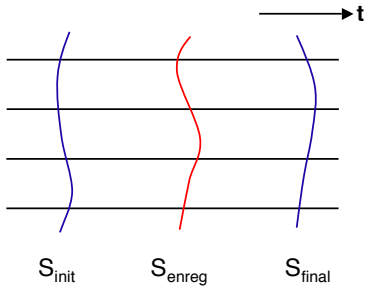
Propriétés stables

- **Un prédicat P sur un état S d'un système est dit stable si**

$$P(S) \text{ et } S \rightsquigarrow S' \Rightarrow P(S')$$

- **Exemples de prédicats stables**
  - **Le système est interbloqué**
  - **Le processus p_i a reçu au moins n messages**
  - **Le calcul est terminé**
  - **L'objet O est inaccessible**
- **Exemple de prédicats non stables**
  - **Le délai sur le routeur R est inférieur à 50 s**
  - **Il y a plus de 100 usagers connectés sur les machines M1 et M2**
  - **x ≥ 2y (x, y variables sur les processus p_i, p_j**

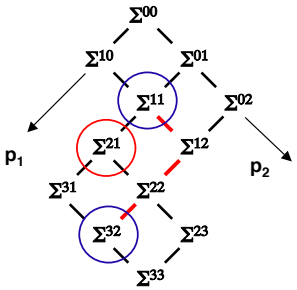
Détecter les propriétés stables



Si P est un prédicat stable, alors :

$$P(S_{enreg}) \Rightarrow P(S_{final})$$
$$\neg P(S_{enreg}) \Rightarrow \neg P(S_{init})$$

Cela est vrai même si l'exécution réelle n'a pas traversé S_{enreg}



Propriétés instables (1)

- **Motivations**
  - Pourquoi considérer des propriétés non stables ?
  - Principal intérêt : mise au point d'un sytème réparti
  - On cherche à vérifier si une propriété a été vraie au cours de l'exécution du système
- **Problème**
  - Incertitude sur la séquence d'états effectivement traversée
- **Méthode**
  - Étant donné une exécution d'un système, on définit des propriétés globales caractérisant un prédicat P :
    - *possibly P* (Pos P): il existe une observation cohérente du système qui passe par un état dans lequel P est vrai
    - *definitely P* (Def P): toutes les observations cohérentes du système passent par un état dans lequel P est vrai
  - On construit des algorithmes d'observation permettant de déterminer si un prédicat P donné satisfait Pos P ou Def P

Propriétés instables (2)

- **Propriétés de Def et Pos**
  - Si S est un état enregistré (par l'algorithme de Chandy-Lamport), et si P est vrai dans S, on peut seulement dire que Pos P est vrai
  - ¬ Pos P ⇒ Def (¬ P) …
  - … mais pas l'inverse
- **Principe de l'évaluation**
  - Un processus "moniteur" M collecte les états (par réception de messages des autres processus)
  - M s'assure que les états sont cohérents (cf méthode plus loin)
  - Pour évaluer Pos P : M parcourt le treillis en partant de l'état initial et s'arrête au premier état pour lequel P est vrai
  - Pour évaluer Def P : M parcourt le treillis niveau par niveau ; tant qu'il existe un chemin ne contenant que des états où P est faux, on ne peut pas conclure Def P. Si aucun tel chemin n'existe pour un niveau donné, alors Def P
  - Explosion combinatoire ! on peut réduire le coût en synchrone

Enregistrement asynchrone (non coordonné) (1)

L'**enregistrement coordonné** (exemple : Chandy-Lamport, ou Koo et Toueg, cf bibliographie) garantit la cohérence de l'état enregistré, mais entraîne un surcoût à l'exécution

- échange de messages supplémentaires
- coût de synchronisation (opérations retardées sur les processus)

Les méthodes non coordonnées (**asynchrones**) visent à réduire ce coût

Deux classes de méthodes selon le traitement des messages reçus

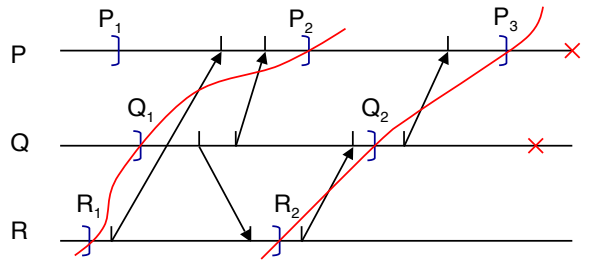
- **méthodes pessimistes**. Tout message arrivant est enregistré sur un support permanent avant d'être traité.
- **méthodes optimistes**. Un message reçu n'est pas immédiatement enregistré (l'exécution peut se poursuivre). Les messages sont enregistrés périodiquement, par paquets

La reprise est plus rapide (pas de perte de messages) pour les méthodes pessimistes, mais l'exécution est ralentie ; c'est l'inverse pour les méthodes optimistes

Enregistrement asynchrone (non coordonné) (2)

L'enregistrement se fait indépendamment pour chaque processus. Les problèmes ne sont détectés qu'au moment de la reprise.

Exemple 1 : processus orphelins
Un processus est dit orphelin s'il a reçu un message dont le départ n'a pas été enregistré dans l'état courant. Cet état est donc incohérent.



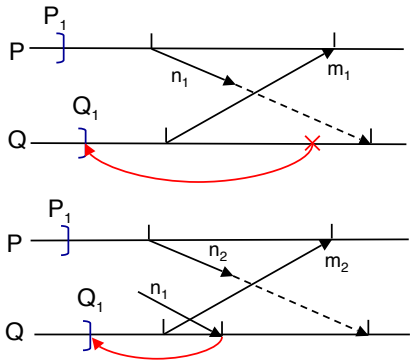
ex : P_3, Q_2, R_2
P_2, Q_1, R_1

peut conduire à l'effet domino

Enregistrement asynchrone (non coordonné) (3)

Exemple 2 : boucle sans progrès (*livelock*) [repris de Koo & Toueg 87]

Répétition indéfinie d'une séquence d'événements, sans progrès possible.



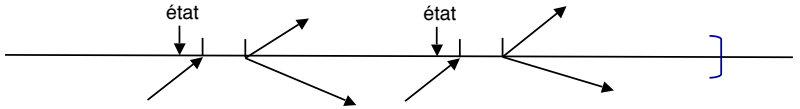
Défaillance de Q : retour à Q_1, donc retour de P à P_1 (pour m_1)

Reprise de Q : réception de n_1, retour de Q à Q_1, donc retour de P à P_1 (pour m_2)

Exemple d'enregistrement asynchrone (1)

Principe : Détecter les processus "orphelins" en comptant les messages envoyés et reçus
Hypothèses :
canaux fiables FIFO, asynchrones
application pilotée par les événements : réception, traitement, émission

Algorithme optimiste : pour chaque processus, on enregistre pour chaque événement, en mémoire volatile : (état, message reçu, messages émis). Périodiquement, ces enregistrements sont copiés en mémoire permanente.



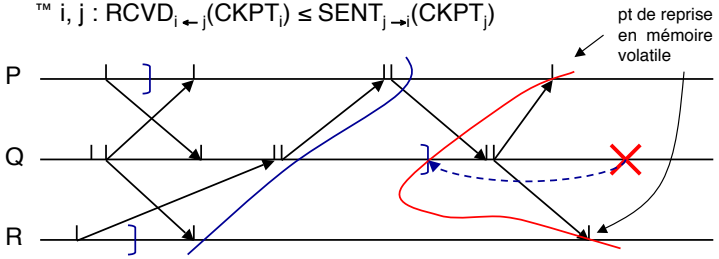
T. Juang, S. Venkatesan. Crash Recovery with little Overhead, *Proc. 11th Int. Conf. On Distributed Computing Systems (ICDCS)*, May 1991, pp. 454-461

Exemple d'enregistrement asynchrone (2)

Soit $RCVD_{i \leftarrow j}(CKPT_i)$ le nb de messages reçus par i depuis j, enregistrés au point de reprise $CKPT_i$
Soit $SENT_{i \rightarrow j}(CKPT_i)$ le nb de messages envoyé par i vers j, enregistrés au point de reprise $CKPT_i$

Alors pour qu'un ensemble de points de reprise $CKPT_i$ soit valide, il faut que :

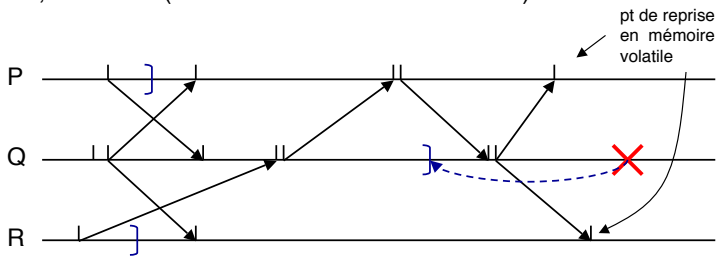
™ i, j : $RCVD_{i \leftarrow j}(CKPT_i) \leq SENT_{j \rightarrow i}(CKPT_j)$



T. Juang, S. Venkatesan. Crash Recovery with little Overhead, *Proc. 11th Int. Conf. On Distributed Computing Systems (ICDCS)*, May 1991, pp. 454-461

Exemple d'enregistrement asynchrone (3)

Algorithme : au moment de la reprise, le processus qui redémarre diffuse un message de reprise à tous. Chacun vérifie que son dernier point de reprise (en mémoire volatile pour les processus non en panne) vérifie la condition de validité. Sinon, il doit remonter au dernier point vérifiant la condition, et réitérer (à cause des réactions en chaîne)



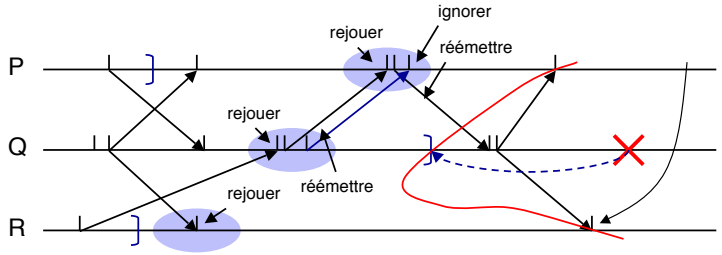
T. Juang, S. Venkatesan. Crash Recovery with little Overhead, *Proc. 11th Int. Conf. On Distributed Computing Systems (ICDCS)*, May 1991, pp. 454-461

Exemple d'enregistrement asynchrone (4)

Lors de la reprise : utiliser les informations enregistrées au point de reprise (état, message reçu, message émis)

rejouer le dernier message reçu (qui a été enregistré)
rejouer le traitement
réémettre les messages émis

ignorer les messages dupliqués



T. Juang, S. Venkatesan. Crash Recovery with little Overhead, *Proc. 11th Int. Conf. On Distributed Computing Systems (ICDCS)*, May 1991, pp. 454-461

Retour sur la datation des événements

- **Systèmes de datation multi-composants**
  - **Horloges vectorielles**
  - **Horloges matricielles**
- **Gestion du temps physique**
  - **Synchronisation des horloges physiques**

Datation des événements

- **Objectif**
  - **Les horloges logiques ont la propriété de validité faible**

$$e \rightarrow e' \Rightarrow HL(e) < HL(e')$$

    - **Nous en avons vu les limitations**
  - **On cherche à caractériser la dépendance causale, en construisant un système de datation H qui ait la propriété de validité forte :**

$$e \rightarrow e' \Leftrightarrow H(e) < H(e')$$

  - **Applications**
    - **observation, mise au point (attribution d'une cause à un effet)**
    - **communication causale, diffusion**
    - **contrôle et maintien de la cohérence d'informations (par exemple pour la reprise)**

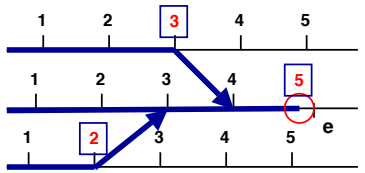
Caractériser la dépendance causale

Rappel : définition du passé (ou historique) d'un événement

$hist(e) = \{e' \mid e' \rightarrow e\} \cup \{e\}$
$hist_i(e) = \{e' \mid e' \rightarrow e \text{ et } e' \in p_i\}$
$hist(e) = \cup hist_i(e), \{e\}$

Dépendance causale : $e' \rightarrow e \equiv e' \in hist(e)$

Idée : inclure le passé de l'événement "émission" dans l'estampille du message



En fait, $hist_i(e)$ est entièrement définie par son événement le plus récent. Il suffit de noter le numéro d'ordre de cet événement

L'histoire d'un événement est ainsi définie par un vecteur de n éléments comprenant les numéros des événements les plus récents de son passé sur chaque processus

k plus grand indice tel que $e_i^k \in hist_i(e)$. Alors pour tout j<k, $e_i^j \in hist(e)$. Donc k définit $hist_i(e)$

Horloges vectorielles (1)

Origine : Fidge, Mattern (1988)

Définition : à tout événement e est associé un vecteur HV(e), comportant un élément par site j :

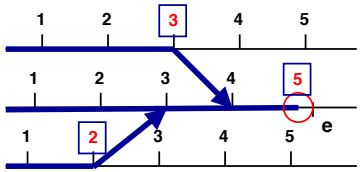
HV(e)[j] = nombre d'événements de $hist_j(e)$ (passé de e dans p_j)

Si e ∈ p_i, alors :

HV(e)[j] = nombre d'événements exécutés dans p_j dont p_i a connaissance au moment de e (connaissance croisée)

ΣHV(e)[j] — 1 = nombre d'événements strictement antérieurs à e

HV(e)[i] = nombre d'événements exécutés par p_i avant e (e compris)



HV(e) = (3, 5, 2)

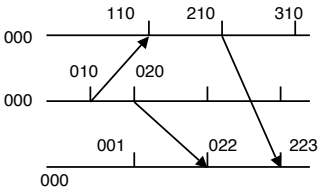
Horloges vectorielles (2)

On entretient un compteur HV_i (horloge locale) sur chaque site i. Un événement e ∈ p_i est daté par HV(e) = valeur courante de HV_i

Règles de mise à jour des horloges sur p_i (analogues à celles des HL)

- valeur initiale : (000)
- événement local sur p_i : $HV_i[i] = HV_i[i] + 1$
- émission d'un message m sur p_i : $HV_i[i] = HV_i[i] + 1$
  le message est estampillé par $E_m = HV_i$
- réception d'un message (m, E_m)
  - $HV_i[i] = HV_i[i] + 1$
  - $HV_i[j] = \max(HV_i[j], E_m[j])$, $j \neq i$

aucun composant de HV_i ne doit être "en retard" par rapport aux messages reçus



Propriétés des horloges vectorielles

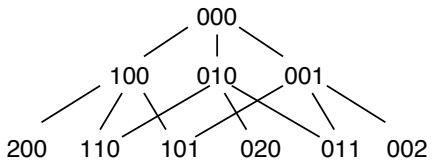
Relation d'ordre. Par définition :

$HV \leq HV' \equiv \forall i, HV[i] \leq HV'[i]$

$HV < HV' \equiv HV \leq HV'$ et $HV \neq HV'$

$HV \parallel HV' \equiv \neg (HV < HV')$ et $\neg (HV' < HV)$

Structure de treillis (induite par les règles de progression)



Horloges vectorielles et dépendance causale

∀a, b	$a \rightarrow b \Leftrightarrow HV(a) < HV(b)$	
$a \neq b$	$a \parallel b \Leftrightarrow HV(a) \parallel HV(b)$	

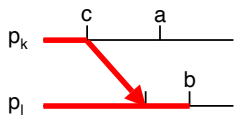
donc

$a \rightarrow b \Leftrightarrow HV(a) < HV(b)$
condition forte de validité

Soit $a \rightarrow b$. Donc $hist(a) \subset hist(b)$, d'où $\forall i\ hist_i(a) \subseteq hist_i(b)$, ce qui s'écrit :

$\forall i\ \{x \in p_i, x \rightarrow a\} \subseteq \{x \in p_i, x \rightarrow b\}$, donc $\forall i\ HV(a)[i] \leq HV(b)[i]$
donc si $a \neq b$, $HV(a) < HV(b)$

Soit $a \parallel b$, $a \in p_k$, $b \in p_l$, $k \neq l$.



c = max ($hist_k(b)$), au sens de →
$c \rightarrow b$ car $c \in hist(b)$ et $c \neq b$ car sur processus différents
$c \rightarrow a$ sur p_k, car si $a \rightarrow c$, alors $a \rightarrow c \rightarrow b$, contradiction avec $a \parallel b$
donc $hist_k(b) = hist_k(c) \subset hist_k(a)$, d'où $HV(b)[k] < HV(a)[k]$

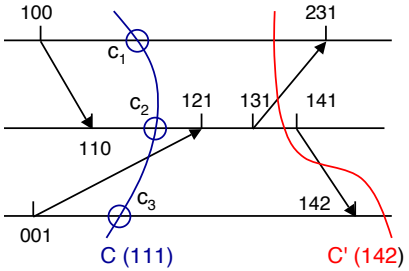
Mais on peut montrer de même que $HV(a)[l] < HV(b)[l]$. Donc $HV(a) \parallel HV(b)$

Horloges vectorielles et coupures cohérentes

Date d'une coupure C = (c_1, …c_n)
définie par HV(C) = sup(HV(c_1), … HV(c_n))
(calculé composant par composant)

$C_1 \subset C_2 \Leftrightarrow HV(C_1) < HV(C_2)$

Condition de cohérence
C cohérente ⇔ HV(C) = (HV(c_1)[1], …HV(c_n)[n])



1) Soit C cohérente. Alors ™ j, HV(c_i)[i] ≥ HV(c_j)[i]

   En effet, l'incrémentation de HV(c_i)[i] ne peut venir que d'un événement local ou résulter d'un message provenant du passé (composant i de l'estampille plus ancien, donc ≤)

2) Soit C non cohérente. Alors ∃ i, j tels qu'un message de p_i a été reçu par p_j avant C et émis après. Soit E(m) l'estampille de ce message. Alors
   HV(c_i)[i] < E(m)[i] ≤ HV(c_j)[i]
   Donc HV(C) > (HV(c_1)[1], …HV(c_n)[n]) (∃ j tel que HV(c_j)[i] > HV(c_i)[i])

Applications des horloges vectorielles

- **Observation d'un système réparti**
  - **datation des événements**
  - **mise au point répartie**
- **Calcul d'état global**
- **Simulation répartie**
- **Diffusion cohérente**

Détection des événements "manquants"

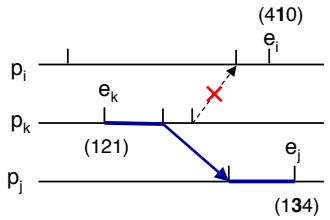
Rappel : si H(e) < H(e'), existe-t-il e" tel que e → e" → e' ?

Pas de réponse avec les horloges logiques

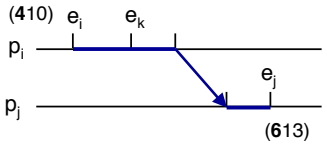
Réponse partielle avec les horloges vectorielles : détection faible

Si $e_i \in p_i$, $e_j \in p_j$ et si ∃ k ≠ j tel que HV(e_i)[k] < HV(e_j)[k], alors

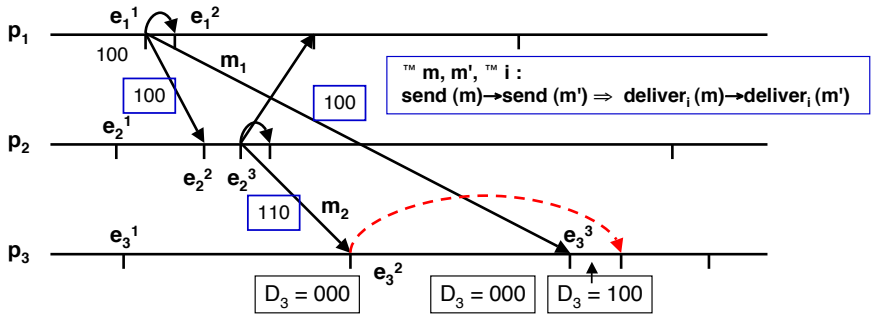
∃ e_k tel que ¬(e_k → e_i) et (e_k → e_j)



Si k = i, alors on a une réponse précise :
si HV(e_i)[i] < HV(e_j)[i], alors ∃ e_k : e_i → e_k → e_j



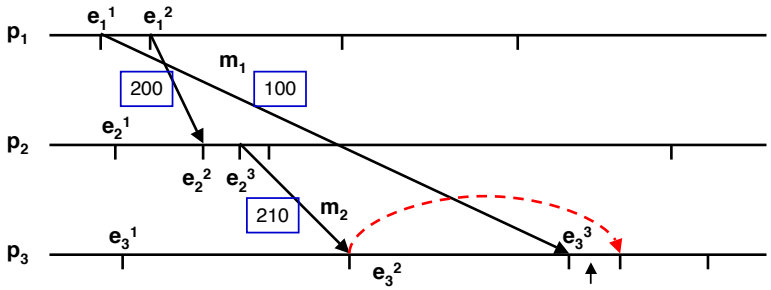
Application : diffusion causale



On utilise des HV restreintes, incrémentées seulement pour l'émission
Chaque processus p_i maintient un tableau D_i[1…n] (n nb de processus)
$D_i[j] = E_m[j]$ où m est le dernier message venant de p_j délivré à p_i

Règle : délivrer (m, E_m) venant de p_j que si $D_i[j] = E_m[j] - 1$ et $D_i[k] \geq E_m[k]$, $k \neq j$

Communication point à point causale



Dans le cas de la communication point à point (par opposition à la diffusion), on ne peut pas détecter l'incohérence (par ex. si m_1 était à destination de p_2 (par ex. vers e_2^1, la situation serait correcte mais les deux cas seraient indiscernables en e_3^2)

La différence entre la communication point à point et la diffusion est que cette dernière implique une **connaissance partagée** (si un processus reçoit un message, il "sait" que tous les autres doivent aussi le recevoir)

Horloges matricielles (1)

- **Des systèmes de datation de plus en plus précis**
  - **Horloges scalaires : HL_i = ce que p_i connaît du système, réduit à un nombre**
  - **Horloges vectorielles : $HV_i[j]$ = ce que p_i connaît de p_j**
  - **Horloges matricielles : $HM_i[j, k]$ = ce que p_i connaît de ce que p_j connaît de p_k**
- **Définition**

On maintient sur chaque site une horloge matricielle (n x n) HM_i

Un événement $e_i \in p_i$ est daté par la valeur courante de HM_i

Un message est estampillé par la valeur courante de HM_i (après mise à jour)

$HM_i[j, k]$ = nombre de messages issus de p_j vers p_k dont p_i a connaissance (i.e. dont l'envoi est causalement antérieur à l'instant présent)

si j = k, il s'agit d'événements locaux au site j

et en particulier : $HM_i[i, i]$: événements locaux à p_i

Horloges matricielles (2)

Règles d'incrémentation sur le site p_i

événement local à p_i : $HM_i[i, i] = HM_i[i, i] + 1$

émission de m vers p_j : $HM_i[i, i] = HM_i[i, i] + 1$
$HM_i[i, j] = HM_i[i, j] + 1$
le message est estampillé par $E_m = HM_i$

réception de (m, E_m) depuis p_j :

On peut maintenant contrôler la délivrance causale

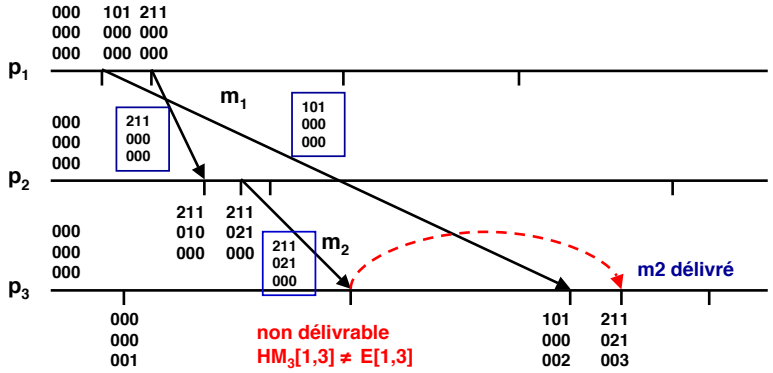
Le message ne peut être délivré que si tous les messages qui lui sont causalement antérieurs ont été délivrés

1. $E_m[j, i] = HM_i[j, i] + 1$ (ordre FIFO sur canal $j \rightarrow i$)
2. Pour tout $k \neq i, j$: $E_m[k, i] \leq HM_i[k, i]$ (messages des autres sites)

On peut alors délivrer le message, et mettre à jour les horloges

1. $HM_i[i, i] = HM_i[i, i] + 1$
2. $HM_i[j, i] = HM_i[j, i] + 1$
3. pour tout $k \neq i, j$ et tout $l \neq i$: $HM_i[k, l] = \max(HM_i[k,l], E_m[k,l])$

Horloges matricielles (3)



Cette méthode assure la délivrance causale des messages point à point

Horloges matricielles (4)

- **Applications**
  - mise au point fine (exemple : noyau Chorus)
  - délivrance causale des messages (exemple : bus A3 Scalagent)
  - gestion de bases de données dupliquées
- **Optimisations**
  - les horloges matricielles sont coûteuses ($O(n^2)$)
  - Diverses optimisations ont été proposées
    - partitionner le système en sous-systèmes (interaction faible entre sous-systèmes) - réduit n
    - exploiter des caractéristiques spécifiques du système (synchronisation)
    - restreindre la "profondeur" du passé

Gestion du temps physique

Les horloges physiques sont utilisées pour construire les horloges internes des processus dans un système d'exploitation. Pour un processus p sur un site : $C(t) = a\ H_i(t) + b$, où $H_i(t)$ est l'indication de l'horloge de S_i au temps t

Propriétés des horloges physiques

Déviation, ou écart (*skew*) au temps t : différence entre deux horloges (par ex. sur deux machines d'un réseau)

Précision : déviation par rapport au temps réel (universel), défini plus loin

Dérive (*drift*) : divergence (déviation croissante) entre horloges due à des fréquences différentes (notamment divergence entre une horloge et l'horloge "idéale" donnant l'heure exacte)

Taux de dérive (*drift rate*) : mesure de la dérive entre deux horloges (en déviation par seconde). Par exemple, une dérive de 1 μs/s conduit à une déviation de 1 s après 12 jours sur deux horloges initialement synchronisées.

Horloges à quartz de haute précision : taux de dérive de 10^{-7} à 10^{-8} s/s

Qu'est-ce que le "temps réel" ?

- **Une convention internationale définit le temps réel**
  - UTC : *Coordinated Universal Time*
  - Utilise une horloge atomique (taux de dérive 10^{-13} s/s) ; ajustement périodique par rapport au temps astronomique
  - L'heure universelle est diffusée par radio et satellites (GPS)
  - Précision de l'heure diffusée
    - 0,1 à 10 ms pour stations terrestres
    - 1 μs pour GPS
  - Un ordinateur muni d'un récepteur peut donc synchroniser périodiquement son horloge interne, et fonctionner comme serveur de temps sur un réseau local

Synchronisation d'horloges physiques

Synchronisation externe

À partir d'une source de temps S (serveur), un ensemble d'horloges H_i est synchronisé de telle sorte que dans tout intervalle Δt de temps réel :

$|S(t) - H_i(t)| < D$ (écart borné) pour tout $t \in \Delta t$

Synchronisation interne

Les horloges H_i sont mutuellement synchronisées, de sorte que dans tout intervalle Δt de temps réel :

$|H_i(t) - H_j(t)| < D$ pour tout couple i, j et pour tout $t \in \Delta t$

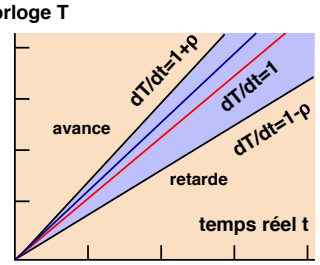
Synchronisation interne n'implique pas synchronisation externe : l'ensemble des horloges peut collectivement dériver par rapport à un serveur

En revanche, la synchronisation externe avec une borne D implique une synchronisation interne avec une borne 2D

Qualités des horloges physiques

Une horloge physique est **correcte** si son taux de dérive (par rapport au temps réel) reste dans des limites prescrites ($\pm\rho$)
L'erreur commise par une telle horloge H sur un intervalle de temps $\Delta t = t' - t$ est bornée :

$(1-\rho)\ \Delta t \le H(t') - H(t) \le (1+\rho)\ \Delta t$



Une horloge correcte au sens ci-dessus n'est pas nécessairement **précise**

$t + \varepsilon \le H(t) \le t + \varepsilon$ Il faut une resynchronisation périodique

Pour certaines applications, il suffit de garantir la **monotonie**

$t' > t \Rightarrow H(t') > H(t)$

Exemple : bon fonctionnement de *make* et des outils analogues

Synchronisation externe : algorithme de Cristian (1)

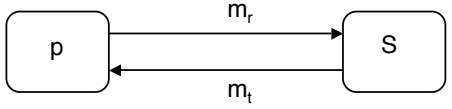
Problème : synchroniser un ensemble d'horloges à parti d'un serveur de temps (lui-même alimenté par le temps universel UTC)

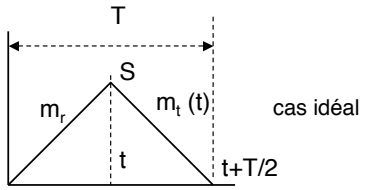
Difficulté : tenir compte du temps de transfert, a priori inconnu

Solution : mesurer le temps d'aller-retour

p demande l'heure à S (message m_r) et reçoit t (message m_t)
Par ailleurs p a mesuré le temps T d'aller-retour de p à S

p règle son horloge au temps t + T/2





Synchronisation externe : algorithme de Cristian (2)

Précision de l'algorithme :

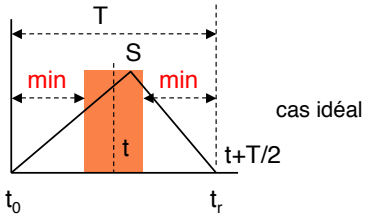
Soit min une borne inférieure du temps de transmission. Alors la précision (écart max. entre l'horloge de p et l'horloge de S) est T/2 — min

En effet, soit t0 l'instant de l'envoi de la requête et tr l'instant de réception de la réponse (dans le temps de S)
Alors

$t0 + min \le t \le t0 + T - min$, donc t dans un intervalle T — 2 min

en supposant que le temps d'aller-retour est effectivement T

Pour la tolérance aux fautes : utiliser plusieurs serveurs synchronisés



Synchronisation interne : algorithme de Berkeley

Problème : synchronisation interne d'un groupe de machines

- Un coordinateur demande aux participants (par diffusion) la valeur courante de leur horloge
- Le coordinateur estime l'heure locale de chaque participant au moyen du temps d'aller-retour (cf Cristian), et calcule la moyenne t_m de ces heures
- Le coordinateur demande aux participants de régler leur horloge sur cette heure moyenne (un participant en avance ne retarde pas son horloge, mais la ralentit pour atteindre progressivement la valeur fixée)

La précision du protocole dépend de l'estimation du temps d'aller-retour. L'idéal est qu'il soit le même pour tous ; en pratique, on fixe un délai maximal et on ne tient pas compte des sites qui dépassent ce délai

Panne du coordinateur : on élit un autre coordinateur (cf plus loin : élection)
Le coordinateur élimine les valeurs trop déviantes (comportement erratique)

NTP (*Network Time Protocol*)

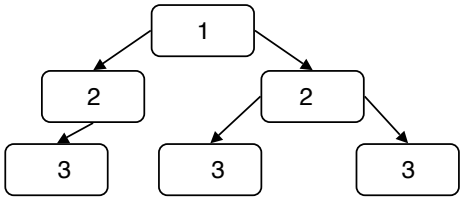
Service de temps de l'Internet

Propriétés

- Fiabilité (redondance des serveurs et des chemins)
- Passage à l'échelle (décentralisation)
- Sécurité (authentification des sources)

3 niveaux

- Serveurs primaires reliés à des sources UTC
- Serveurs secondaires synchronisés sur les serveurs primaires
- Serveurs tertiaires (chez les usagers)



NTP : synchronisation des serveurs

Reconfiguration dynamique en cas de panne

- un serveur primaire qui perd la connexion à lasource UTC peut devenir serveur secondaire
- un serveur secondaire qui perd la connexion vers son primaire peut contacter un autre primaire

Modes de synchronisation

- **Multicast**.Diffusion de l'heure sur un réseau local (niveau 3). Les clients se mettent à l'heure en estimant le délai de propagation. Faible précision
- **RPC**. Les clients interrrogent un seveur de niveau 3 (algorithme de Cristian). Meilleure précision
- **Pair à pair** (utilisé aux niveaux 1 et 2). Des couples de serveurs se synchronisent mutuellement (cf plus loin). Précision élevée

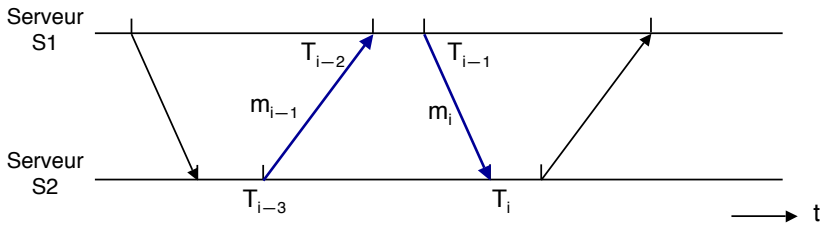
NTP : synchronisation pair à pair

Communication via UDP (protocole léger)

Chaque message porte une estampille :

- heures locales d'envoi et de réception du message précédent
- heure locale d'envoi du message courant

Le récepteur possède donc les heures d'envoi et de réception des deux derniers messages (T_{i-3}, T_{i-2}, T_{i-1}, T_i)



NTP : estimation de la précision

Pour tout couple de messages (m_{i-1} et m_i) échangés entre deux serveurs, on estime un écart e_i entre leurs deux horloges et un délai total de transmission $t_{i-1} + t_i = T_{i-2} - T_{i-3} + T_i - T_{i-1}$

Soit e l'écart réel de S_2 par rapport à S_1. Soit t_{i-1} et t_i les temps de transmission des messages m_{i-1} et m_i

$T_{i-2} - T_{i-3} = t_{i-1} + e$, et $T_i - T_{i-1} = t_i - e$

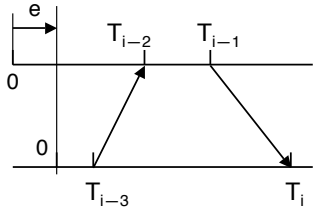
D'où par soustraction :

$e = e_i + (t_i - t_{i-1})/2$, avec

$e_i = (T_{i-2} - T_{i-3} - (T_i - T_{i-1}))/2$

donc puisque t_{i-1} et $t_i > 0$ et $t_{i-1} + t_i = d_i$:

$e_i - d_i/2 \le e \le e_i + d_i/2$



on a donc un écart estimé avec une précision connue