Algorithmique Distribuée
Communication de groupes

Laurent PHILIPPE

Master 2 Informatique
UFR des Sciences et Techniques

2013/2014

Les groupes de processus

Définition

- Ensemble de processus vu comme une seule abstraction
- Donner une vision commune
- Rendre un service en commun
- Partager une ressource

Propriétés

- Dynamique vs. Statique
- Groupe de pairs vs. Groupe hiérarchisé
- Ouvert vs. Fermé

La communication de groupe

Communication externe

- Accès au groupe plutôt qu'à un membre
- Redirection interne :
  - Tolérance aux pannes
  - Équilibrage de charge
  - Travail à plusieurs

Communication interne

- Point-à-point
- Diffusion : ordre de réception des messages

Rappel

Hypothèses de travail

- Temps de communication finis mais non prévisibles
- Exécution d'un processus est une suite d'évènements
- Évènements internes ou externes
- Pas de panne des processus (machines)
- Messages arrivent intacts
- Messages ne sont pas dupliqués

Sommaire

Sommaire

Rappel : relation de précédence

Définition

- Définit un ordre dans les événements d'un système distribué
- Entre :
  - Deux évènements dans un même processus
  - Émission d'un message par un processus
  - Réception d'un message par un processus
- La relation de précédence est notée : $\rightarrow$
- L'expression $E_1 \rightarrow E_2$, se lit l'événement E_1 précède l'événement E_2, c'est à dire que E_1 a lieu avant E_2

Rappel : relation de précédence

Propriétés

La relation de précédence détermine un ordre partiel entre les évènements :

- Transitivité : si $E_1 \rightarrow E_2$ et $E_2 \rightarrow E_3$ alors $E_1 \rightarrow E_3$
- Irréflexibilité : il est pas possible d'avoir $E_1 \rightarrow E_1$
- Antisymétrie : si $(E_1 \rightarrow E_2)$ alors $\neg(E_2 \rightarrow E_1)$

On définit la propriété de concurrence entre deux évènements :

- E_1 et E_2 sont concurrents si ils ne sont pas liés par une relation de précédence : $\neg(E_1 \rightarrow E_2) \bigwedge \neg(E_2 \rightarrow E_1)$

Relation de causalité

Définition

- Un événement E_1 produit un événement E_2 si et seulement si E_2 est une conséquence de E_1.
- Par exemple :
  - l'émission d'un message produit la réception de celui-ci
  - une instruction peut être produite par une autre
  - la réception d'un message peut produire une émission

Notation

- La relation de causalité est notée $\preceq$
- L'expression $E_1 \preceq E_2$, se lit l'événement E_1 produit l'événement E_2, c'est à dire que E_2 est la conséquence de E_2

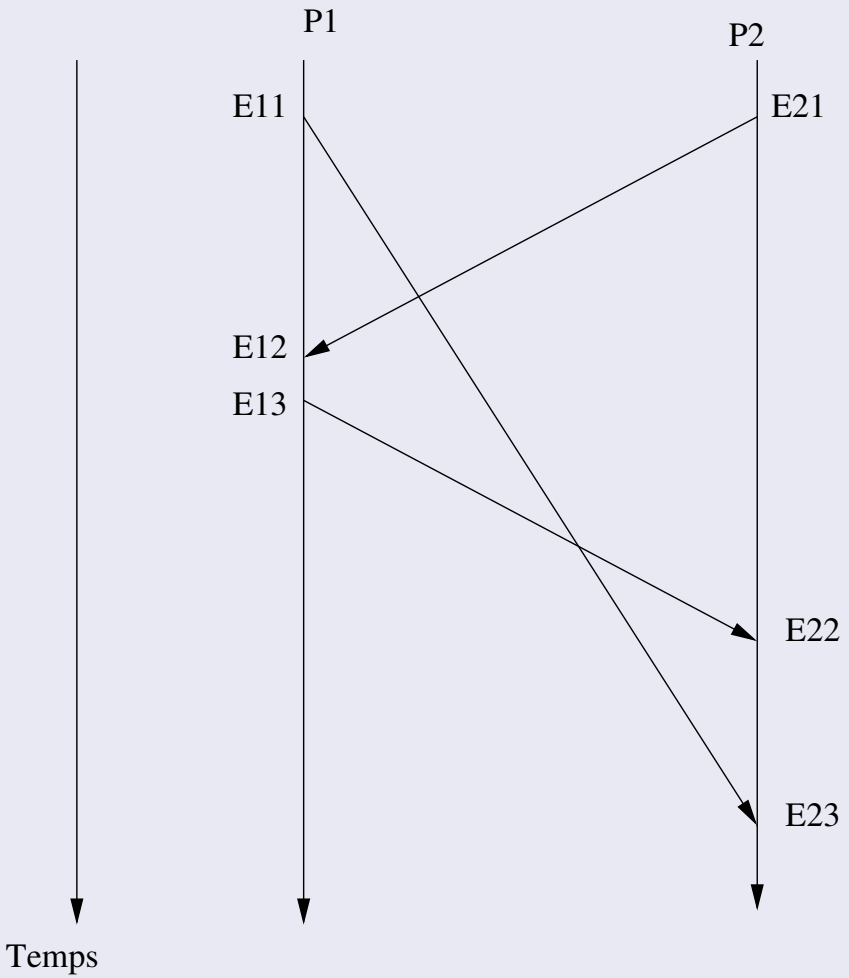
Relation de causalité

Propriétés

La relation de causalité détermine un ordre partiel entre les évènements :

- transitivité : si $E_1 \preceq E_2$ et $E_2 \preceq E_3$ alors $E_1 \preceq E_3$
- irréflexibilité : on n'a pas $E_1 \preceq E_1$
- antisymétrie : si $(E_1 \preceq E_2)$ alors $\neg(E_2 \preceq E_1)$

Exemple de relation de causalité



- $E_{11} \preceq E_{23}$
- $E_{13} \preceq E_{22}$
- $E_{21} \preceq E_{12}$
- Les événements suivants sont concurrents : $E_{11} \parallel E_{21}$

Remarque estampilles

Estampilles

- Horloges logiques de Lamport
- Estampille définit un ordre total
- Tous les événements peuvent être ordonnés les uns par rapport aux autres
- Ordre arbitraire entre estampilles égales sans risque pour la causalité, choix basé sur le numéro de site

Ordre Causal

Définition

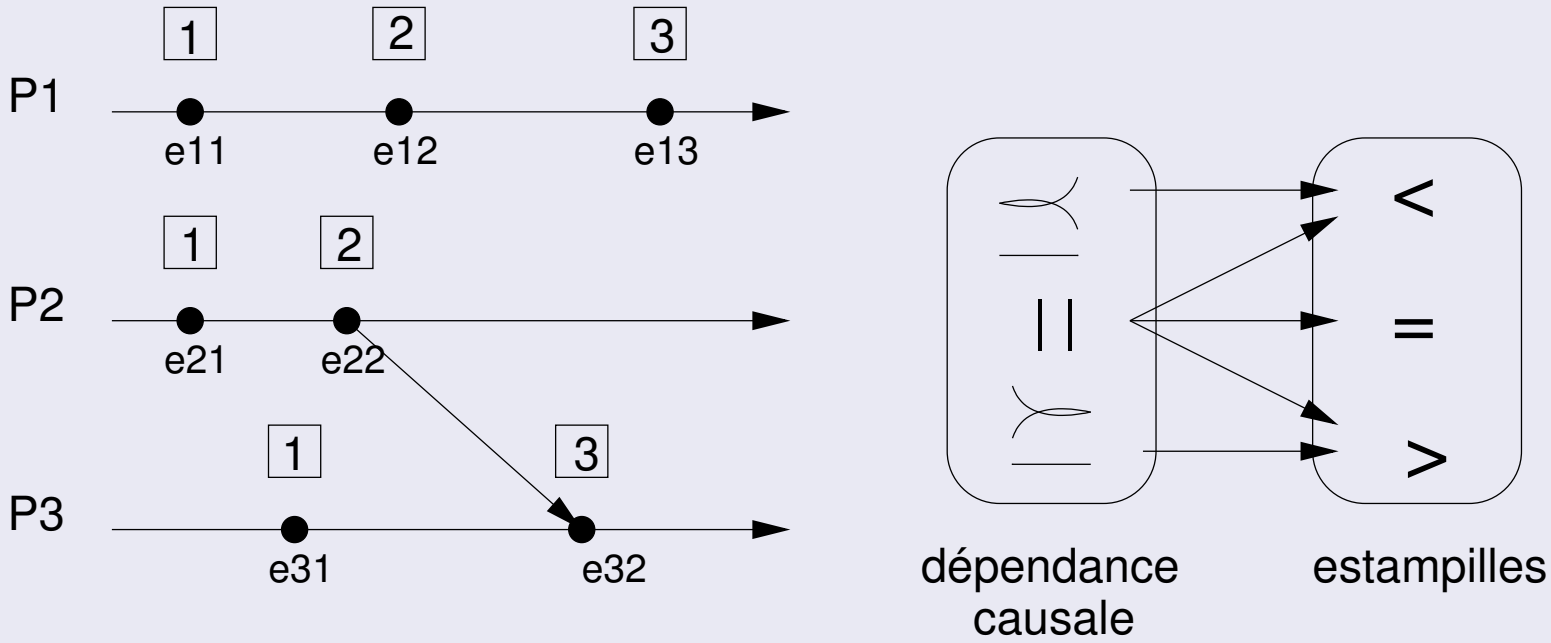
- Dépendance causale est un ordre partiel
- dépendance causale entre deux événements e_a et e_b, notés $e_a \preceq e_b$ signifie :
  - e_a précède e_b sur le même processus (ordre séquentiel) ;
  - e_a correspond à l'envoi du message reçu en e_b ;
  - $\exists$ un événement e_c tel que $e_a \preceq e_c$ et $e_c \preceq e_b$.

Chemin causal

Définition et propriétés

- Ensemble des événements entre les événements *causes* et les événements *conséquences*
- Si $e_a \preceq e_b$, alors il existe au moins un chemin causal entre e_a et e_b
- Deux événements concurrents ne sont pas liés par une dépendance causale et il n'y a pas de chemin qui les relie
- L'ordre des estampilles est alors quelconque
- Connaissance des chemins de causalité ne permet pas toujours d'ordonner tous les événements

Interaction indépendance causale ↔ estampille



Interaction indépendance causale $\leftrightarrow$ estampille

- Estampilles effacent artificiellement la notion de dépendance causale
- Restent cohérentes avec la dépendance causale
- Dans certains cas, l'ordre total est inutile : on ne cherche que la précédence
- Les estampilles ne sont pas *denses* :
  si $H(e_1) < H(e_2)$, on ne peut pas savoir s'il existe e_b tel que $e_1 \preceq e_b$ et/ou $e_b \preceq e_2$.

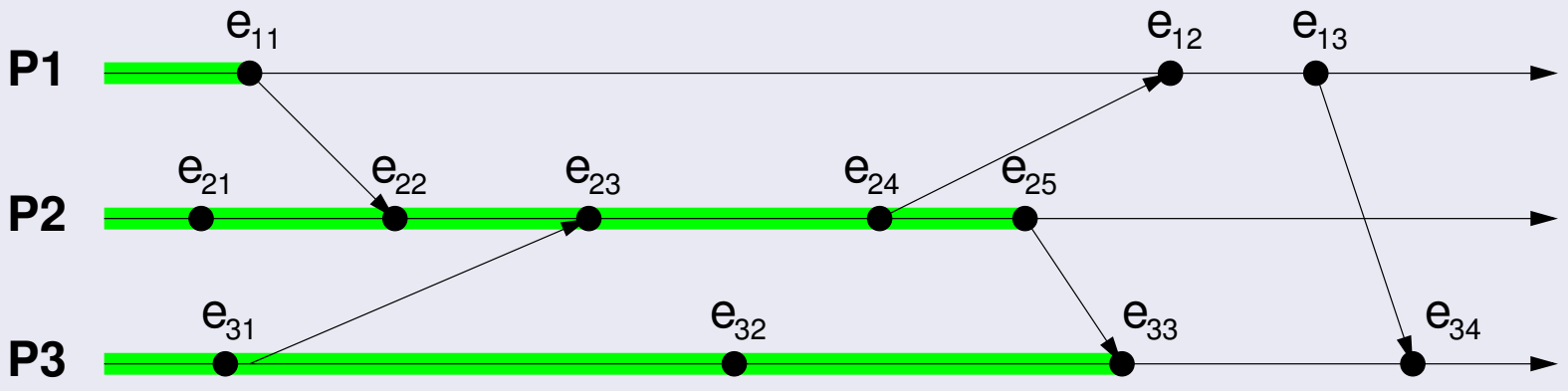
Les historiques

Définition

- Nécessaire de conserver l'historique pour la causalité
- On définit par $hist(e)$ l'historique d'un événement e
- $hist(e)$ = ensemble des événements appartenant à tous les chemins causals
- $hist(e) = \{e\} \cup \{e' \mid e' \preceq e\}$

Les historiques

Exécution d'événements sur 3 processus



$$hist(e_{33}) = \{e_{11}, e_{21}, e_{22}, e_{23}, e_{24}, e_{25}, e_{31}, e_{32}, e_{33}\}$$

Les historiques

- Historique d'un événement e est utilisé pour la datation de e
- Examen de son passé permet de déterminer ses dépendances causales
- Dépendance de deux événements e et e' :

$$\begin{cases} hist(e) & = & \{e\} \cup \{e' \mid e' \preceq e\} \\ e \preceq e' & \Leftrightarrow & e \in hist(e') \\ e \parallel e' & \Leftrightarrow & (e \notin hist(e')) \wedge (e' \notin hist(e)) \end{cases}$$

- Inconvénient de l'historique : sa taille croît sans cesse

Projection de l'historique

Définition

- projection de l'historique $hist(e)$ d'un événement e sur le processeur P_i est l'ensemble :

$$hist_i(e) = \{e' \in hist(e) \mid e' \in P_i\}$$

- Sur la figure :

$$hist(e_{33}) = \{e_{11}, e_{21}, e_{22}, e_{23}, e_{24}, e_{25}, e_{31}, e_{32}, e_{33}\}$$

Les projections de $hist(e_{33})$, respectivement sur les processeurs P_1, P_2 et P_3, sont :

- $hist_1(e_{33}) = \{e_{11}\}$
- $hist_2(e_{33}) = \{e_{21}, e_{22}, e_{23}, e_{24}, e_{25}\}$
- $hist_3(e_{33}) = \{e_{31}, e_{32}, e_{33}\}$

Projection de l'historique

Propriété

Si $e_{i,k}$ événement de projection de l'historique de e sur P_i, alors tout événement antérieur à $e_{i,k}$ sur P_i est dans l'historique de e :

$$\exists k \mid e_{i,k} \in hist_i(e) \Longrightarrow \forall j < k : e_{i,j} \in hist(e)$$

Réduction de la représentation de l'historique

- Soit l'événement $e_{i,k}$, le plus récent des événements de l'historique de e sur P_i
- D'après la propriété, $\forall e_{i,k-j} \mid k > j > 0, e_{i,k-j} \in hist_i(e)$
- Il suffit de ne conserver que l'entier k pour caractériser la projection de l'historique de e sur P_i.
- Horloges vectorielles : un événement par site

Représentation d'un historique par un vecteur

- $hist(e) = \bigcup_i hist_i(e)$: un vecteur $V(e)$, avec une ligne par processeur, suffit pour représenter $hist(e)$
- Pour un système de n processus, le vecteur est défini de la manière suivante :

$$\forall i (1 \leq i \leq n) : V(e)[i] = k, \text{ tel que :}$$

$$e_{i,k} \in hist_i(e) \text{ et } e_{i,k+1} \notin hist_i(e)$$

- $V(e)[i]$: nombre d'événements de P_i connus de e

Exemple

Exemple précédent

Dans l'exemple précédent on a :

- $V(e_{33})[1] = 1$
- $V(e_{33})[2] = 5$
- $V(e_{33})[3] = 3$

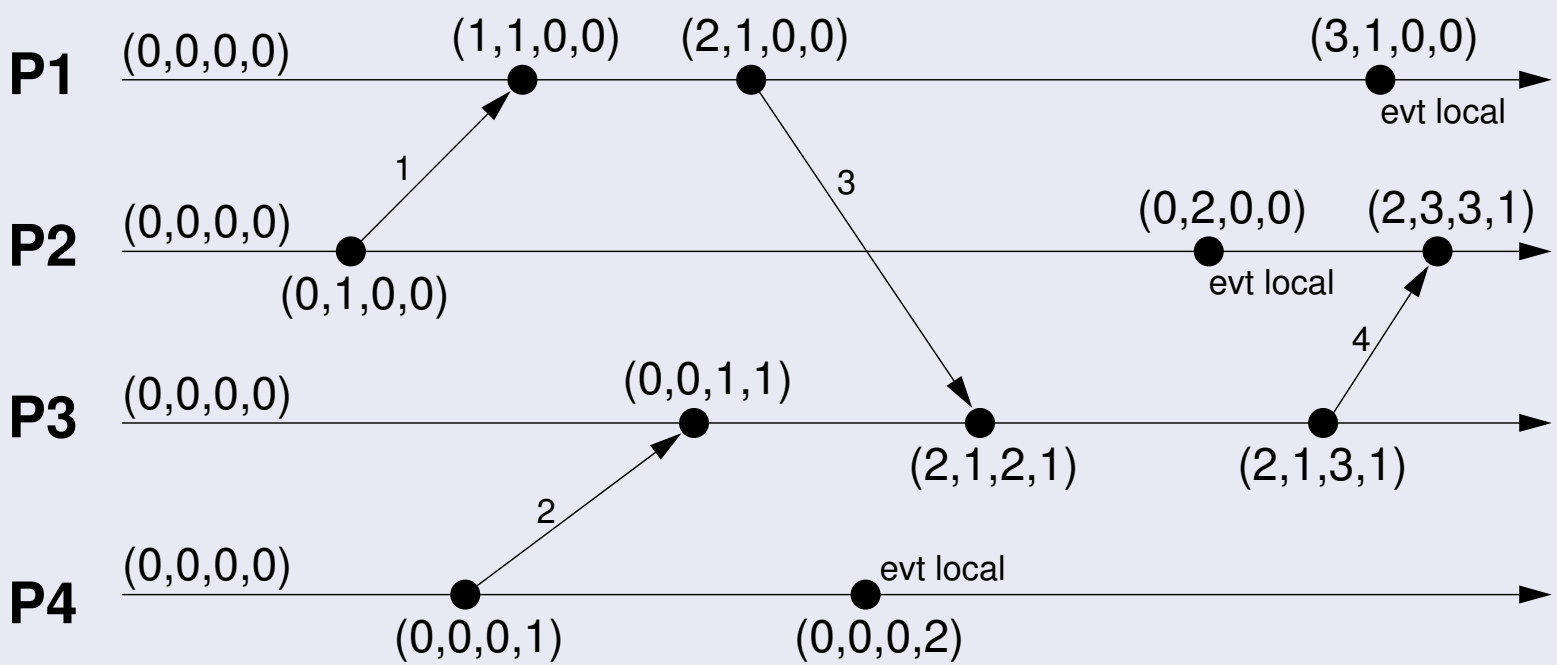
Les horloges vectorielles (Fidge, Mattern 1988)

Définition

- Un vecteur V_i de taille n est associé à chaque processus P_i
- Initialement $V_i = (0, \ldots, 0)$
- A chaque événement local à P_i, $V_i[i] \leftarrow V_i[i] + 1$
- A chaque émission, le message m porte l'estampille V_e (V_e est l'horloge vectorielle de l'émetteur P_e)
- A la réception de (m, V_e) par un processus P_i :
  - $V_i[i] \leftarrow V_i[i] + 1$
  - $V_i[j] \leftarrow max(V_i[j], V_e[j])$ pour $j = 1, \ldots, n, j \neq i$

Exemple d'exécution

Séquence d'événements sur 4 processeurs



Les horloges vectorielles

Propriété : ordre partiel

Soit deux horloges vectorielles V et V', il existe un ordre *partiel* entre ces 2 horloges :

$$V \leq V' \text{ si et seulement si } \forall i(1 \leq i \leq n) : V[i] \leq V'[i]$$

$$V < V' \text{ si et seulement si } (V \leq V') \wedge (V \neq V')$$

$$V \| V' \text{ si et seulement si } \neg(V < V') \wedge \neg(V' < V)$$

Les horloges vectorielles

Propriété : denses

- Soient e_i un événement du processeur P_i et e_j un événement du processeur P_j.
- Si $V(e_i)[k] < V(e_j)[k]$ avec $k \neq j$, alors il existe e_k tel que $\neg(e_k \rightarrow e_i) \wedge (e_k \rightarrow e_j)$.

Signification

Il existe forcément un événement e_k qui a permis l'incrémentation de la composante k de l'horloge sur le processeur P_k. Cet événement a eu lieu avant l'événement e_j sans qu'on puisse dire s'il s'est produit avant ou après l'événement e_i. La seule chose que l'on puisse affirmer, c'est que e_k n'est pas la cause de e_i.

Sommaire

Diffusion fiable

Propriété, définition

Pour être fiable, une diffusion doit satisfaire 3 critères :

- **validité** : lorsqu'un processus diffuse, tous les membres du groupe reçoivent ;
- **accord** : si un processus reçoit, alors tous les autres membres du groupe reçoivent ;
- **intégrité** : chaque message n'arrive qu'une et une seule fois.

Diffusion fiable

Différents types de diffusions fiables

- FIFO : les messages sont délivrés dans le même ordre que l'ordre d'envoi, protocole FBCAST
- Causale : les messages sont délivrés dans un ordre compatible avec le respect de la causalité, protocole CBCAST ;
- Atomique : les messages sont tous délivrés dans le même ordre, protocole ABCAST.

Diffusion fiable

Mise en œuvre

Dans le contexte des protocoles de diffusion fiable

- Faire la distinction entre *réception* d'un message et *délivrance* de ce message
- Le système n'a aucun contrôle sur le temps de communication des messages, ni sur la date d'arrivée ou de réception des messages
- Différer la délivrance d'un message au processus destinataire afin de se conformer aux définitions précédentes
- Ne favorise pas les performances du système, mais garantit le bon fonctionnement d'une application distribuée.

Les protocoles de diffusion fiable

Notations

- **receive**$_p(m)$: réception du message m par le processus p ;
- **deliver**$_p(m)$: livraison du message m au processus P

Propriété

Dans tous les cas et de manière évidente, la réception précède toujours la délivrance du message. On note :

$$\textbf{receive}_p(m) \rightarrow \textbf{deliver}_p(m)$$

Ordre FIFO : FBCAST

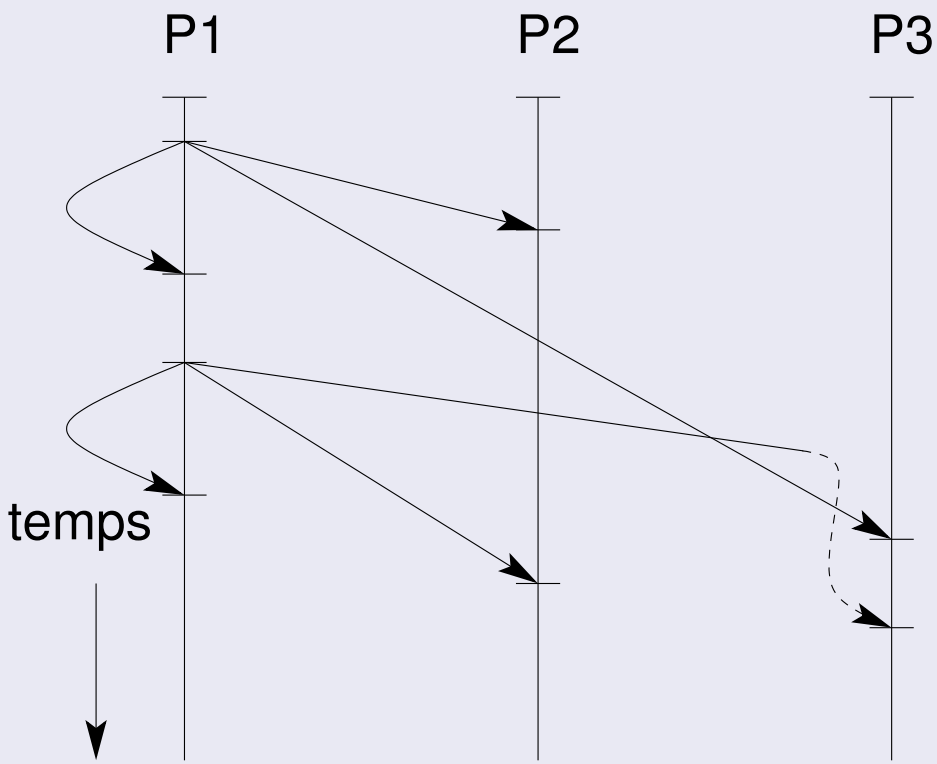
Définition

Si un processus diffuse un message $m1$ avant de diffuser un message $m2$, alors aucun processus correct ne délivre $m2$ à moins qu'il n'ait déjà délivré $m1$.
Les messages envoyés par un processus p sont délivrés dans l'ordre d'émission par tout processus q correct :

Si $\exists m, m'$ et $p \in g$ |
$\text{send}_p(m) \rightarrow \text{send}_p(m') \Longrightarrow \forall q \in g : \text{deliver}_q(m) \rightarrow \text{deliver}_q(m')$

Ordre FIFO : FBCAST

Exécution correcte d'une diffusion fiable de type FIFO



Ordre causal : CBCAST

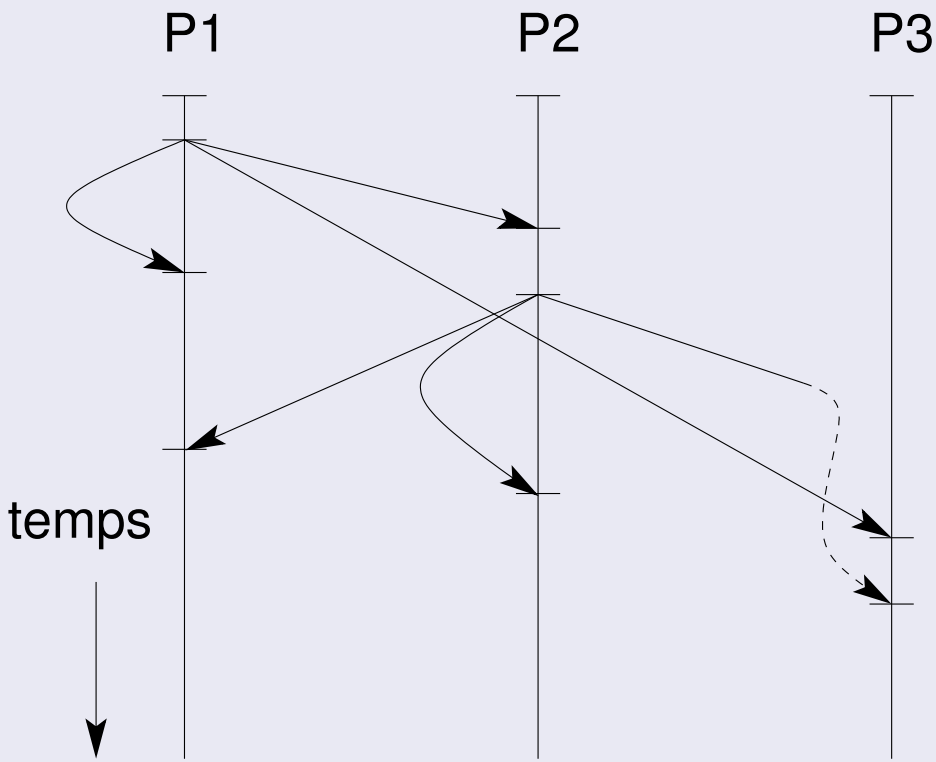
Définition

Si le message m est la cause du message m' (m' est envoyé après la délivrance de m sur le processus émetteur de m') alors tous les processus délivrent le message m' après le message m. Ceci peut s'écrire de la manière suivante :

$$\text{Si } m \rightarrow m' \Longrightarrow \forall p \in g \text{ alors } \text{deliver}_p(m) \rightarrow \text{deliver}_p(m')$$

L'ordre de livraison des messages respecte la relation de causalité entre m et m'.

Ordre causal : CBCAST



Ordre atomique ou total : ABCAST

Définition

La relation d'ordre est étendue aux processus concurrents. En effet, si on a un groupe g, tous les processus p de ce groupe délivrent les messages dans le même ordre, quelque soit l'ordre l'émission des messages, d'où l'expression :

$\exists m, m'$ et $p \in g \mid$
$\text{deliver}_p(m) \rightarrow \text{deliver}_p(m') \Rightarrow \forall q \in g : \text{deliver}_q(m) \rightarrow \text{deliver}_q(m')$

Ordre atomique ou total : ABCAST

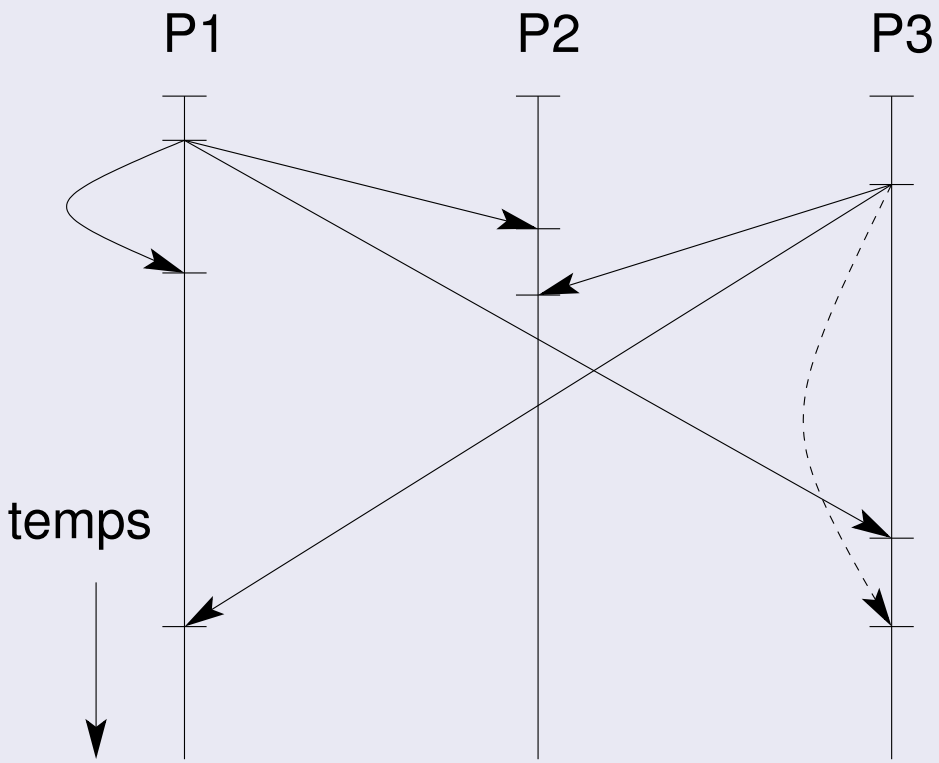


Figure: Exemple d'exécution correcte d'une diffusion fiable atomique

Protocoles de diffusion fiable

Application au multi-cast

- Diffusion par sous-groupes
- Définition de groupes différents
- Diffusion fiables sur plusieurs groupes
- Relations s'appliquent aux intersections
- Pas de contraintes en dehors des intersections

Sommaire

Protocole FBCAST

Protocole

- Le message m est accompagné du numéro de séquence d'émission $\#seq(m)$ et du numéro du processus émetteur $sender(m)$
- Sur le processus émetteur, les numéros de séquence d'émission constituent une suite continue (sans trou)
- Alors chaque processus p connaît le prochain numéro de séquence $next(q)$ du message à délivrer en provenance de chaque processus q du groupe g
- Grâce à ces informations, le processus p récepteur d'un message m est capable de délivrer ou non le message, et s'il le délivre, il essaie de délivrer tous les messages en provenance de q qui n'auraient pas encore été délivrés

Protocole FBCAST

Protocole

- Initialement, les numéros de séquence des messages en provenance de tous les autres processus ont la valeur 1, sur tous les processus du groupe
- À la réception, sur p, d'un message m en provenance de q, m est stocké parmi les messages attendant d'être délivrés ;
- Tant qu'il existe un message m en provenance de q tel que le prochain numéro de séquence du message $next(q)$ est égal à $\#seq(m)$, alors le message m est délivré, supprimé de l'ensemble des messages à délivrer et le prochain numéro de séquence du message à délivrer en provenance de q ($next(q)$) est incrémenté.

Protocole FBCAST

Exercices

- Ecrire l'algorithme sous forme de règles
- On dipose d'une file de messages en local (*put*, *get*)

Sommaire

Le protocole CBCAST

CBCAST utilise l'historique :

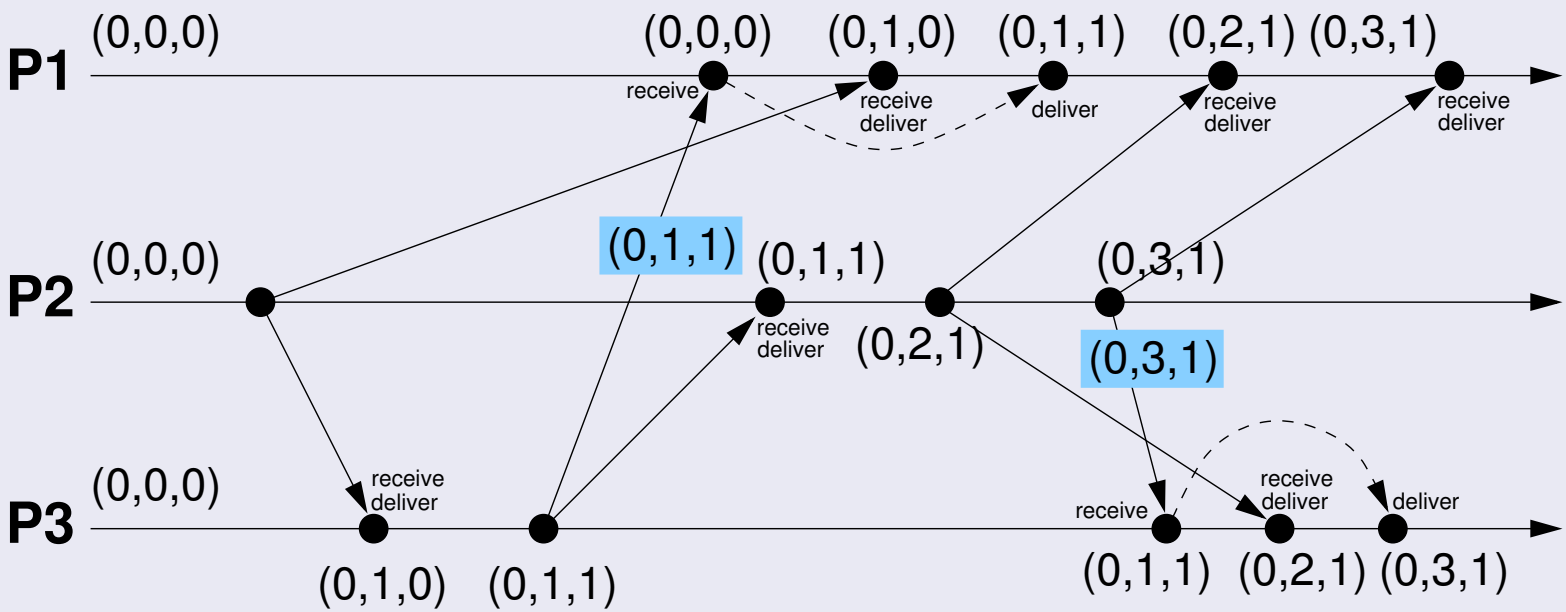
1. Avant d'envoyer m, le processus P_i incrémente $V_i[i]$ et estampille le message m avec V_i ;
2. À la réception d'un message m estampillé par V_m en provenance de P_i, le processus $P_j \neq P_i$ diffère sa livraison jusqu'à ce que les conditions suivantes soient réalisées :

$$\begin{cases} 1.\ V_j[i] = V_m[i] - 1 \\ 2.\ \forall k \in [1, n] \text{ et } k \neq i : V_j[k] \geq V_m[k] \end{cases}$$

   La condition 1. indique que le message qui arrive de i porte le numéro attendu et donc qu'aucun message de i n'a été perdu. Pour le processus P_i, la condition est $V_i[i] = V_m[i]$
3. Après remise de m : $V_j \rightarrow max(V_m, V_j)$

Le protocole ABCAST

Déroulement de diffusions fiables de type CBCAST



Protocole CBCAST

Exercices

- Dérouler l'algorithme pour le cas où :
  1. on a trois processus,
  2. le processus P_0 diffuse, la diffusion se termine
  3. le processus P_1 diffuse,
  4. le processus P_2 diffuse avant d'avoir reçu le message de P_1
  5. P_0 diffuse après avoir reçu le message de P_1,
  6. P_2 reçoit le message de P_0 avant celui de P_1
- Écrire l'algorithme sous forme de règles, en supposant qu'on dispose d'une file de messages en local (*put*, *get*)

Sommaire

Le protocole ABCAST

Principe

- Pour respecter l'ordre total (ou atomique), on utilise un protocole de validation à deux phases.
- Envoie d'abord avec une estampille provisoire,
- Puis calcule l'estampille définitive du message diffusé grâce à un échange entre les processus du groupe et le processus émetteur.
- Le processus émetteur valide ensuite la valeur de l'estampille définitive et la communique aux autres processus destinataires afin que le message puisse être délivré dans l'ordre des estampilles définitives.
- Protocole coûteux prend $3n$ messages.

Le protocole ABCAST

Protocole - 1ère Phase

- Soit g un groupe de n processus P_i avec $i = 1 \ldots n$.
- Lors d'une diffusion le processus p_i émetteur joint à la diffusion du message l'estampille provisoire : $\langle$`date d'émission.numéro de l'émetteur`$\rangle$ et l'envoie au membres du groupe
- A la réception, Chaque destinataire :
  - P_j met sa propre estampille au message reçu : état courant de l'estampille + 1
  - Marque le message en attente : *Pending*

Le protocole ABCAST

Protocole - 2nde Phase

1. Chaque destinataire P_j renvoie le message avec l'estampille attribuée à l'expéditeur P_i du message ;
2. Quand l'expéditeur P_i a reçu toutes les réponses, il choisit la plus grande estampille $\langle$`date d'émission.numéro de l'émetteur`$\rangle$ comme estampille définitive puis envoie une validation avec cette estampille à tous les membres du groupe ;
3. A la réception de l'estampille définitive, le message est marqué validé : *Ready* ;
4. Les messages validés sont alors délivrés aux applications *dans l'ordre de leurs estampilles*.

Le protocole ABCAST

Exemple

- Diffusions fiables de type ABCAST au sein d'un groupe de trois processus
- Soit les processus P_1, P_2 et P_3 diffusant chacun respectivement les messages $m1$, $m2$ et $m3$.
- L'état courant des estampilles de ces processus est 14 pour P_1, 15 pour P_2, 16 pour P_3 ;
- $P1$ reçoit les messages dans l'ordre $m3$, $m1$ et $m2$
- P_2 reçoit les messages dans l'ordre $m2$, $m1$ et $m3$
- P_3 reçoit les messages dans l'ordre $m1$, $m3$ et $m2$

Exemple ABCAST

P_1

m3	m1	m2
15.1	**16.1**	17.1
P	P	P

P_2

m2	m1	m3
16.2	**17.2**	18.2
P	P	P

P_3

m1	m3	m2
17.3	18.3	19.3
P	P	P

A la réception des messages sur les processus du groupe, ils sont mis en attente et le protocole leur associe des estampilles provisoires. Tous les processus envoient un message vers le processus P_1 avec l'estampille provisoire du message $m1$.

Exemple ABCAST

P_1

m3	m2	m1
15.1	**17.1**	17.3
P	P	**R**

P_2

m2	m1	m3
16.2	17.3	18.2
P	**R**	P

P_3

m1	m3	m2
17.3	18.3	**19.3**
R	P	P

Le processus P_1 choisit la plus grande estampille ($e = \max\{16.1, 17.2, 17.3\} = 17.3$) comme définitive et la diffuse. A sa réception les messages sont notés prêts (R) à être délivrés. Localement, ils sont réordonnés en fonction des nouvelles estampilles pour être délivrés dès qu'ils auront la plus petite estampille locale. Les processus diffusent maintenant les estampilles provisoires pour le message provenant de P_2

Exemple ABCAST

P_1

m3	m1	m2
15.1	17.3	19.3
P	R	**R**

P_2

m1	m3	m2
17.3	18.2	19.3
R	P	**R**

P_3

m3	m2	
18.3	19.3	
P	**R**	

Le message m1 est délivré sur le processus $P3$. L'estampille définitive pour m2 est trouvée (19.3) et est diffusée à tous les processus.

Exemple ABCAST

P_1

m1	m3	m2
17.3	18.3	19.3
R	**R**	R

P_2

m3	m2	
18.3	19.3	
R	R	

P_3

m3	m2	
18.3	19.3	
R	R	

Le message m1 est délivré sur le processus P_2 grâce au tri en fonction des estampilles. Suite à l'envoi sur P_3 des estampilles provisoires du message m3, le processus P_3 peut choisir l'estampille définitive pour son message (18.3). Après diffusion de cette estampille, tous les messages sont prêts (R) et sont délivrés dans l'ordre de la liste triée des messages, la même pour tous les processus

Protocole ABCAST

Exercices

- Dérouler l'algorithme pour le cas où :
  1. On a quatre processus,
  2. L'état initial est 8 pour P_0, 10 pour P_1, 6 pour P_2 et 7 pour P_3
  3. Le processus P_1 diffuse un message qui est traité par l'ensemble des membres du groupe,
  4. Les processus P_0 et P_3 diffusent ensuite un message en même temps,
  5. le processus P_2 diffuse un message alors que les deux messages précédents ont été reçus mais non validés (fin de première phase).
- Écrire l'algorithme sous forme de règles.

Conclusion

- ISIS
- Ouvrage de référence : Michel Raynal. *La communication et le temps dans les réseaux et les systèmes répartis*, Eyrolles. Collection de la Direction des Études et Recherches d'Électricité de France, CEA-EDF-INRIA, École d'été d'informatique. 1991.
- Sites web de référence :
  - Michel Riveill. *Quelques algorithmes répartis et leur usage pour les systèmes*. `http://rangiroa.essi.fr/cours/systeme2/96-flips-algo-rep2.pdf`. 2003.
  - Denis Conan. *Initiation à l'algorithmique répartie*. `http://www-inf.int-edu.eu/COURS/ASR4/COURS/AlgoRep/EnLigne/Contenu/8.html`. juin 2007.