Au sujet des systèmes répartis

Acadie

30 janvier 2017

plan

Plan

Qu'est-ce que système distribué ?

Caractéristiques physiques

- Ensemble d'entités (sites/processus/pairs) interagissant via un réseau de communication
- Pas d'horloge globale : chaque site a son horloge
- Pas d'état global immédiat accessible à un site
- Fiabilité globale faible : panne partielle
- Parallélisme d'exécution

Modèle d'interaction

Communication par mémoire partagée

Variables accessibles à tous les processus.
Hypothèses sur le comportement en cas d'accès concurrent (registres atomiques,...)

Communication par message

Variables locales (privées) à chaque processus.
Opérations explicites d'envoi et de réception de messages : `send(msg, destinataire)`, `broadcast(msg)`...

Spécifications formelles usuelles

Pseudo-algorithmes

Pseudo-algorithmes, $\pm$ proche des commandes gardées de Dijkstra
Ex : Pluscal

Systèmes de transitions

Systèmes de transitions en intention (variables, actions)
Ex : TLA$^+$, Spin...

Calculs de processus

Opérations explicites de communication
Ex : CSP, CCS, π-calculus

Spécification en syst. de transitions $\rightarrow$ réalité à messages

Observations

- Systèmes de transitions avec états décrits par des variables $\approx$ modèle mémoire partagée (sans accès concurrent).
- Système (réellement) réparti = pas de mémoire partagée, communication par messages

(autres cas : calculateur parallèle à mémoire physiquement partagée ; implantation de mémoire virtuelle répartie)

Spécification en syst. de transition $\rightarrow$ réalité à messages

Bonnes pratiques

- Variables indexés par les processus : $x \in Proc \rightarrow X$
  $+$ des variables d'interaction
- Seul p peut utiliser $x[p]$ $\quad \approx$ variables locales privées
- Règles d'usage des variables d'interaction :
  $net' = net \cup \{m\}$ $\quad \approx$ envoi d'un message
  $m \in net \wedge net' = net \setminus \{m\}$ $\quad \approx$ réception d'un message

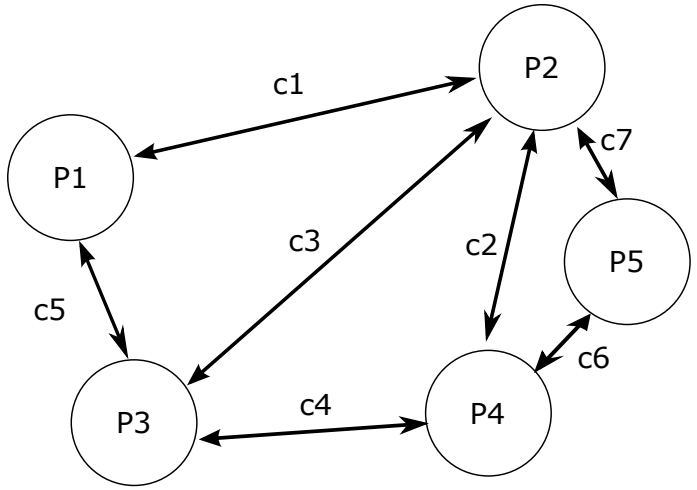
Isoler les opérations de communication

Spécification de `send` et `receive`

Exemple : découvrir le graphe de communication

Le problème

On considère un ensemble de sites connectés 2 à 2 par des canaux bidirectionnels.
Comment chaque site peut-il apprendre la structure du graphe ?



Connaissance initiale

Chaque site connaît son identité (id_i) et l'identité de ses voisins $voisins_i = \{id_x, id_y, \ldots\}$.
Le couple (id_i, id_j) représente le canal entre id_i et id_j (et symétriquement).
Aucun site ne connaît l'identité de tous les sites, ni leur nombre.

Découverte : principe de l'inondation

Le problème redéfini

Comment chaque site id_i peut-il connaître l'ensemble des canaux (id_j, id_k) existants ?

Principe

- Un site i qui veut connaître le graphe diffuse à tous ses voisins sa position $(id_i, voisins_i)$
- Quand un site i reçoit un message $(id_k, voisins_k)$ pour la première fois, il met à jour sa connaissance du graphe et transmet le message à ses voisins. Sinon, il l'ignore.
- Il conclut quand il a reçu un message de tous les sites dont il a eu connaissance via les voisinages

Algorithme pour le site i

```
on start :
  for each id_j ∈ voisins_i do
      send (id_i, voisins_i) to id_j
  end for
  sites_known_i ← {id_i}
```

Variables locales au site i :
id_i : son identité (const)
$voisins_i$: ses voisins (const)
$sites\_known_i$: les sites dont il a reçu un message
$channels\_known_i$: les canaux qu'il a appris

```
on reception (id, voisins) :
  if sites_known_i = ∅ then start(); fi
  if id ∉ sites_known_i then        -- premier message de id
    sites_known_i ← sites_known_i ∪ {id}
    channels_known_i ← channels_known_i ∪ { (id, id_k) : id_k ∈ voisins }
    for each id_j ∈ voisins_i        -- propage le message
        send (id, voisins) to id_j
    end for
    if ∀ (id_j, id_k) ∈ channels_known_i : {id_j, id_k} ⊆ sites_known_i then
        id_i connaît le graphe. FIN
    endif
  endif
```

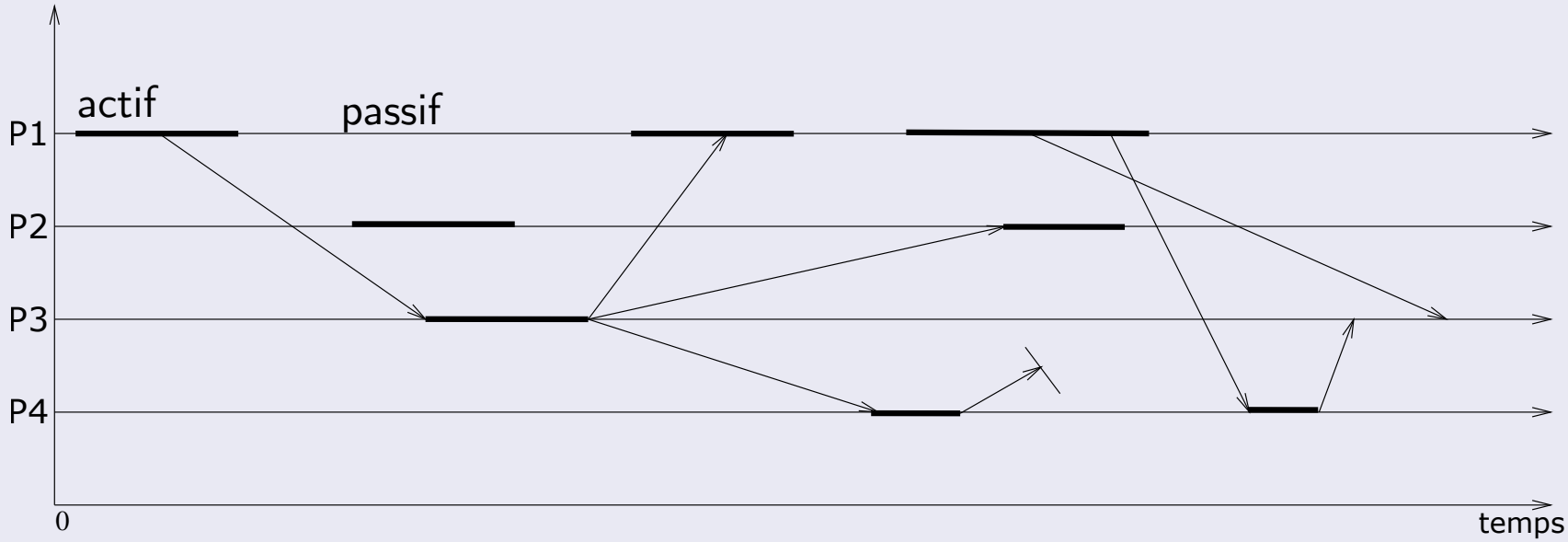
Questions

- Terminaison : tous les sites finissent-ils par atteindre FIN ? (oui)
- Terminaison : un site peut-il savoir que les autres ont terminé ? (non)
- Correction : à la terminaison de i, $channels\_known_i$ = le graphe ? (oui)
- Correction : après terminaison de tous, $\forall i, j : channels\_known_i = channels\_known_j$? (oui)
- Coût en messages ? (2 * nombre de sites * nombre de canaux)
- Complexité en temps ? (2 * diamètre)
  (diamètre = $\max_{(i,j)} \min distance(i, j)$)

Plan

Vision dynamique : chronogramme

Une exécution répartie

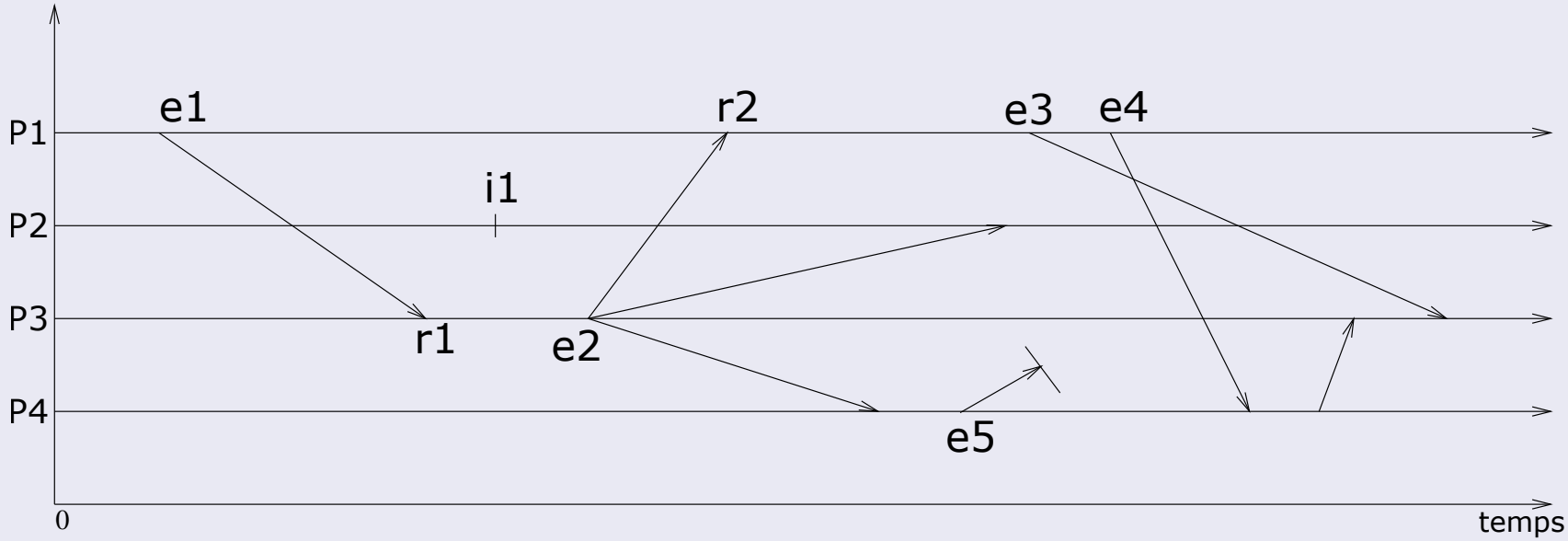


Absence d'état global instantanément observable → actions longues assimilables à des événements : émission, réception, interne

Vision dynamique : Chronogramme

Représentation événementielle

3 types d'événements : émission, réception, interne



Ordre partiel entre les événements = la causalité

Relation de causalité (Lamport 1978)

Ordre partiel entre événements $\prec$

- Les événements d'un processus sont totalement ordonnés : e et e' sur le même site, et e précède e', alors $e \prec e'$.
- L'émission d'un message précède causalement sa réception : Si $e =$ émission(m) et $e' =$ réception(m), alors $e \prec e'$.
- Transitivité : $\forall e, e', e'' : e \prec e' \prec e'' \Rightarrow e \prec e''$

- La relation $\prec$ est un ordre partiel :

$$e \parallel e' \stackrel{\Delta}{=} e \not\prec e' \wedge e' \not\prec e$$

- Indépendance du temps physique mais consistent avec :

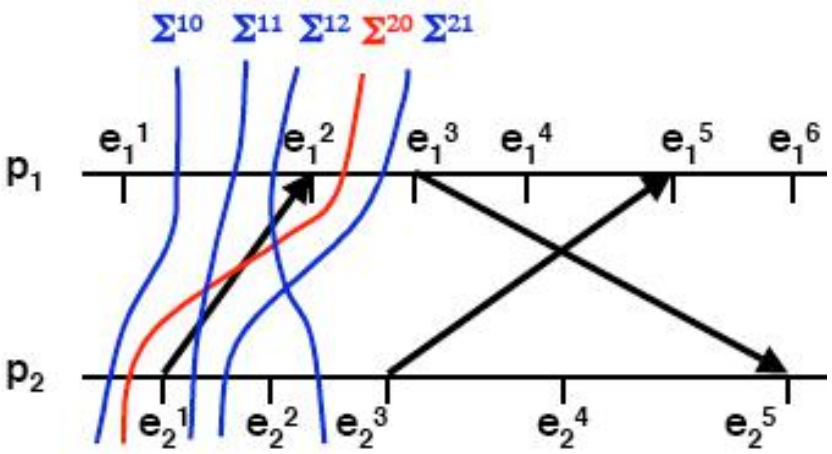
$$e \prec e' \Rightarrow e \text{ est survenu avant } e' \text{ dans le temps absolu}$$

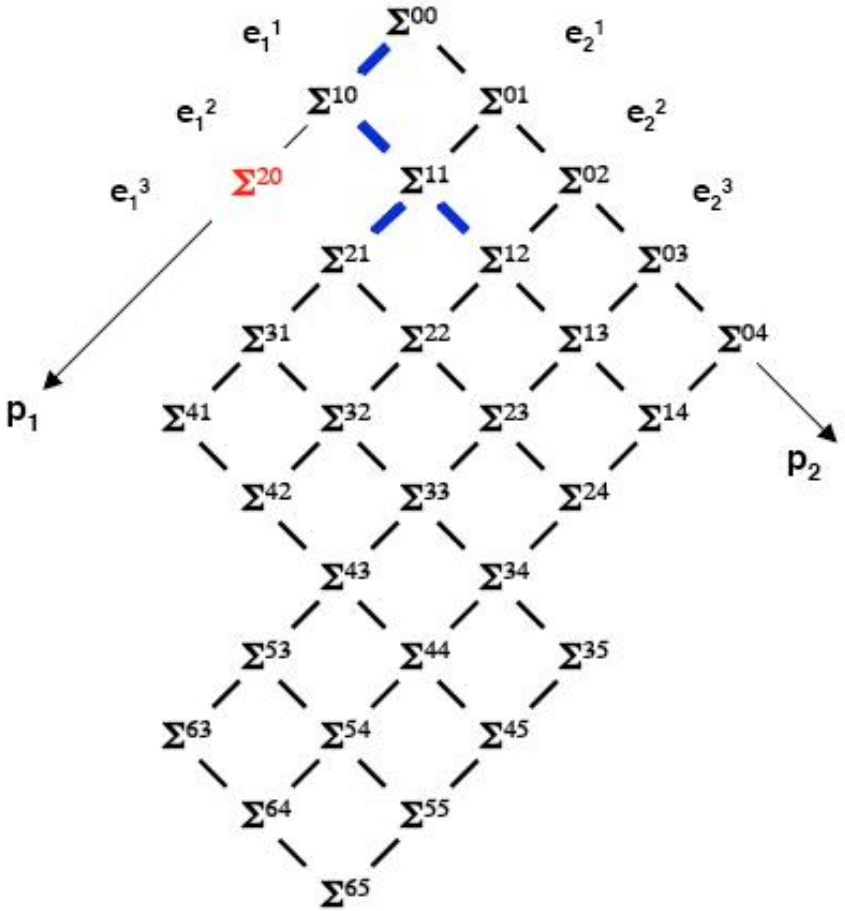
Exécution “physique”

Entrelacement : linéarisation de l'ordre causal
Hypothèse d'absence de simultanéité (ou comme si)
$\Rightarrow$ plusieurs exécutions linéaires causalement équivalentes : $e_1 \parallel e_2$
$\rightsquigarrow \langle \ldots e_1 \ldots e2 \ldots \rangle$ ou $\langle \ldots e_2 \ldots e1 \ldots \rangle$

Event structure (Winksel 1980) = causalité + relation de consistance/conflit

Treillis des coupures cohérentes





- Arc du treillis = occurrence d'un événement possible
- Une exécution = suite d'états globaux cohérents = chemin dans le treillis

Explosion du nombre d'exécutions causalement équivalentes

(dessins : cours S. Krakowiak)

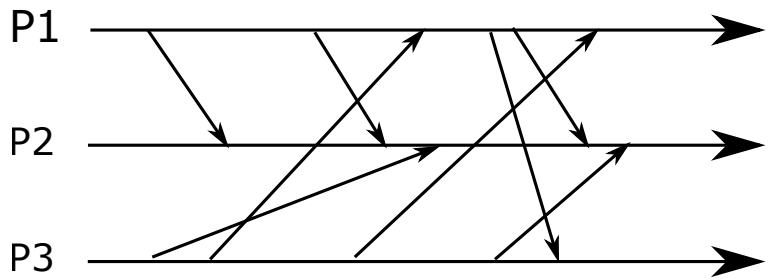
Système asynchrone vs synchrone

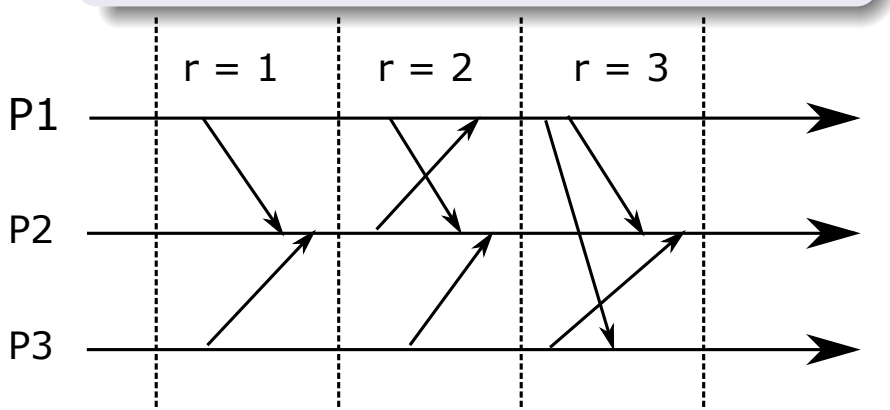
Asynchrone

- Pas de temps externe
- Progression de chaque processus à son rythme
- Délai de transmission arbitraire

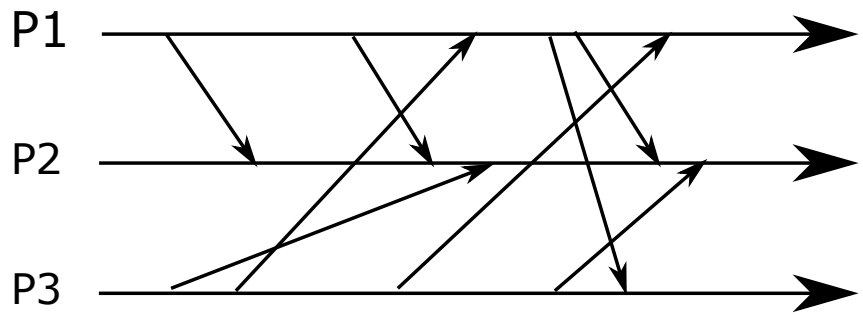
Synchrone

- Existence de pas de calcul (*round*) globaux
- Un message émis dans un pas est reçu au pas suivant / dans le même pas (selon le modèle)





Système asynchrone

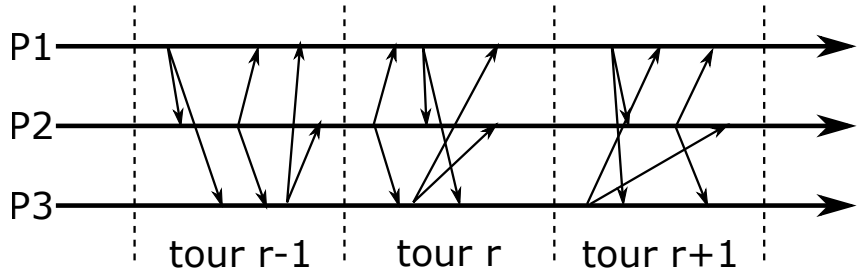


- Pas de temps externe
- Progression arbitraire de chaque processus
- Délai de transmission arbitraire

$\Rightarrow$

- Modèle réaliste
- Nombreux résultats d'impossibilité

Système synchrone



- Existence de pas de calcul (*round*) globaux
- Un message émis dans un pas est reçu au pas suivant / dans le même pas (c'est pareil)

$\Rightarrow$

- Modèle moins réaliste mais approchable avec des horloges synchronisés ou une coordination externe
- Des résultats de possibilité

Variante : rendez-vous (CSP/CCS) : $\approx$ un seul message en transit à la fois. *Modèle identique (paraît-il)*