Cours ASPD
Algorithmes répartis: auto-stabilisation et nommage
Telecom Nancy 2A Apprentissage

Dominique Méry
Telecom Nancy
Université de Lorraine

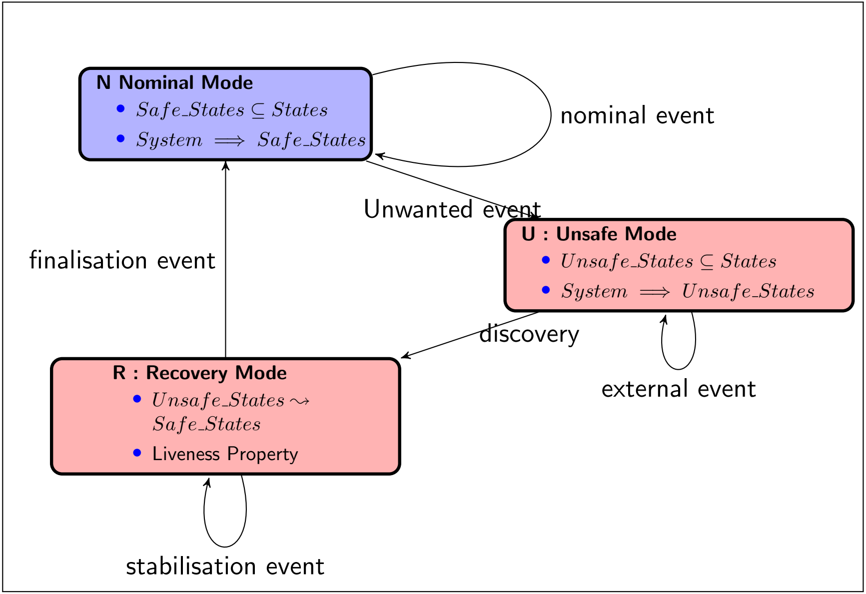
Année universitaire 2024-2025
29 janvier 2026(9:21pm)

Sommaire

Section Courante

Etat courant

Auto-stabilisation dans les systèmes répartis



Stabilisation

- Les systèmes peuvent être victimes de fautes transitoires
- Un certain nombre d'algorithmes permettent d'anticiper de tels problèmes.
- Les algorithmes auto-stabilisants constituent une alternative à la perte de l'état consistant d'un système donné à la suite d'une défaillance.
- Différents phénomènes peuvent apparaître lors de l'exécution d'un algorithme : en particulier peuvent se produire des changements dans le réseau (ajout ou disparition d'un sommet ou d'un lien) ou bien encore des altérations de messages ou de mémoires.
- Un algorithme est dit auto-stabilisant s'il se termine correctement en dépit de l'apparition de ces phénomènes.
- Un algorithme auto-stabilisant ne nécessite pas d'initialisation particulière.

Contexte

- Les systèmes répartis sont exposés à des risques de défaillances transitoires qui peuvent placer un système donné dans un état ou une configuration arbitraire.
- Cette configuration arbitraire peut être une configuration ne satisfaisant plus l'invariant du système.
- Dans ce cas, la question est de concevoir un algorithme réparti permettant de faire converger le système global d'une configuration arbitraire vers une configuration dite stable et satisfaisant l'invariant.

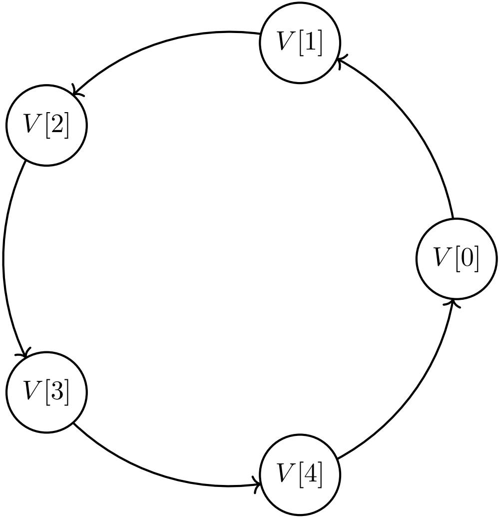
Etat courant

Modélisation d'un système auto-stabilisant

- Soit un système réparti $\mathcal{S}$ modélisé par un système de transition $(States, \longrightarrow)$ où $States$ est un ensemble d'états ou de configurations et où $\longrightarrow$ modélise la relation de transition simulant l'activité du système.
- Une exécution ou un calcul de $\mathcal{S}$ est une suite maximale $(\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_i, \ldots)$ engendrée par $(States, \longrightarrow)$.
- Tout préfixe d'une telle suite est un calcul puisqu'il n'y a pas d'états initiaux déterminés.
- Tout préfixe d'exécution est donc un calcul ou une exécution de $\mathcal{S}$ (pas de dépendance d'un état initial)

Etat courant

Réseau en anneau



Etat courant

Problème de l'exclusion mutuelle

Description

- Dans toute configuration, au plus un processus a le privilège.
- Chaque processus a le privilège infiniment souvent.

- Dans un réseau avec une topologie en anneau, des processus ont accès à une ressource.
- Le privilège est accordé à un processus à la fois.
- Le problème est que le privilège est géré par un jeton qui peut être perdu et donc être dans une configuration ne satisfaisant plus la propriété.

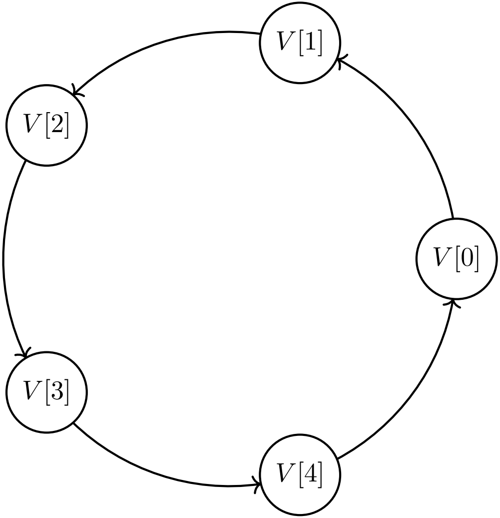
Algorithme de Dijkstra

$$\text{DOMAINE} \triangleq 0..N$$
$$\text{IMAGE} \triangleq 0..M$$

$$\text{NToZero} \triangleq$$
$$\land \quad V[0] = V[N]$$
$$\land \quad V' = [V \ \textbf{EXCEPT} \ ![0] = (V[0]+1)\%(M+1)]$$

$$\text{Others(I)} \triangleq$$
$$\land \quad I \in 0..N$$
$$\land \quad I \neq N$$
$$\land \quad V[I+1] \neq V[I]$$
$$\land \quad V' = [V \ \textbf{EXCEPT} \ ![I+1] = V[I]]$$

Réseau en anneau



Propriétés de stabilisation

```
Init == V = [i \in 0..N |-> (IF i # N THEN i ELSE 0)]
Next == NToZero \/ (\E I \in 0..N-1:Others(I))

Prop1 ==    (V[0] = V[N])
              => (\A i \in 1..N-1: V[i+1] = V[i])

Prop2 ==
      \/  (\E i,j  \in 0..N-1: i # j
              /\ V[i+1] #  V[i] /\ V[j+1] #  V[j])
      \/ (V[0] = V[N] /\
            (\E i  \in 0..N-1:  V[i+1] #  V[i]))
```

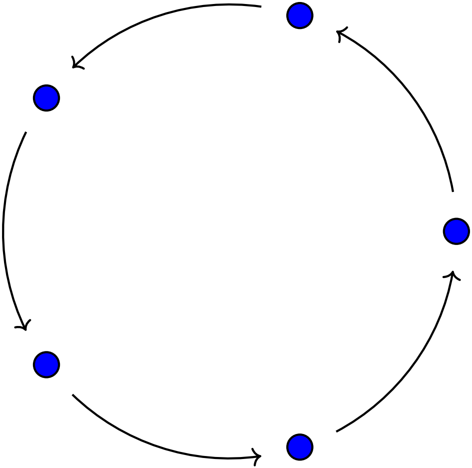
Etat courant

Coloriage d'un anneau

Le problème du coloriage d'un anneau concerne l'existence d'une configuration dans laquelle trois nœuds consécutifs n'ont pas la même couleur et pose la question de définir un algorithme réparti permettant de l'atteindre à parrtir de toute configuration intiale de l'anneau.

- Modéliser un processus de coloration d'un anneau à partir d'une situation quelconque initiale pour atteindre une situation stable caractérisée par le fait que deux nœuds voisins n'ont pas la même couleur.
- La règle de changement de couleur : Si un nœud a la même couleur que l'un de ses vosins, il chosit une autre couleur que celle de ses voisins.
- La règle est applicable tant que deux nœuds ont la même couleur.
- La règle est non-applicable dès que les nœuds adjacents ont tous une couleur différente : le cas d'un anneau à un nœud est exclus en considérant qu'il y a au moins deux nœuds dans l'anneau.

Exemple d'anneau



CONTEXT $ring$
SETS
N
CONSTANTS
n
AXIOMS
$axm1 : n \in N \rightarrowtail\!\!\!\!\twoheadrightarrow N$
$axm2 : \forall s \cdot s \subseteq n[s] \Rightarrow N \subseteq s$
$axm3 : finite(N)$
END
MACHINE $one{-}shot$
SEES $ring, color$
VARIABLES
col
INVARIANTS
$inv1 : col \in N \rightarrow color$

EVENTS
EVENT INITIALISATION
$act1 : col :\in N \rightarrow color$
EVENT stable
ANY
f
WHERE
$grd1 : f \in N \rightarrow color$
$grd2 : \forall x \cdot f(x) \notin f[(n \cup n^{-1})[\{x\}]]$
THEN
$act1 : col := f$
END

Le modèle définit la solution à trouver par un événement abstrait exprimant le lien entre la configuration initiale et une configuration stable. Les régles données explique comment le calcul est effectué par les différents processus du réseau. Chaque règle suppose que leur application est possible quand trois nœuds consécutifs vérifient une condition donnée.

Chaque règle constitue un élément de calcul réparti et cet élément est appliqué sur la *boule* centrée sur un nœud donné et tient compte des couleurs des nœpériphériques.

MACHINE $rule$ **REFINES** $one{-}shot$
SEES $ring, color$
VARIABLES col, c
INVARIANTS
$inv1 : c \in N \rightarrow color$
THEOREMS
$thm1 :$

$$\left(\begin{array}{l} \forall x \cdot c(x) \neq c(n(x)) \\ \vee \\ \exists x, clr \cdot c(x) = c(n(x)) \wedge clr \neq c(x) \wedge clr \neq c(n(x))) \end{array} \right)$$

EVENTS

EVENT INITIALISATION

$$act1 : col, c : | \left(\begin{array}{l} col' \in N \rightarrow color \\ \wedge \\ c' \in N \rightarrow color \\ \wedge \\ col' = c' \end{array} \right)$$

EVENT stable REFINES stable

WHEN

$grd1 : \forall x \cdot c(x) \neq c(n(x))$

WITNESSES

$f : f = c$

THEN

$act1 : col := c$

END

EVENT rule

STATUS **convergent**

ANY

x, clr

WHERE

$grd1 : c(x) = c(n(x)) \vee c(x) = c(n^{-1}(x))$

$grd2 : clr \neq c(n(x))$

$grd3 : clr \neq c(n^{-1}(x))$

THEN

$act1 : c(x) := clr$

END

La terminaison de ce processus dépend de la condition appelée variant et qui doit décroître par application des règles.

$$\boxed{\textbf{VARIANT}\ card(\{x|c(x) = c(n(x))\})}$$

TLA$^+$

Les graphes VISIDIA sont symétriques ; le graphe g est défini à partir du graphe n et la règle $rule$ est raffinée en $visidia{-}rule$

CONTEXT $ringgr$
EXTENDS $ring1$
CONSTANTS
g
AXIOMS
$axm1 : g = n \cup n^{-1}$
END

EVENT INITIALISATION

$act1 : col, c : |(col' \in N \to color \wedge c' \in N \to color \wedge col' = c')$

EVENT stable REFINES stable

WHEN

$grd1 : \forall x \cdot c(x) \notin c[g[\{x\}]]$

THEN

$act1 : col := c$

END

EVENT rule

STATUS convergent

REFINES rule

ANY

x, clr

WHERE

$grd1 : c(x) \in c[g[\{x\}]]$

$grd2 : clr \notin c[g[\{x\}]]$

THEN

$act1 : c(x) := clr$

END

Dérivation d'un programme VISIDIA

Rule : $\overset{cl1}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\overset{cl1}{\bullet}$ et $\overset{cl1}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\overset{cl2}{\bullet}$ se réduit en $\overset{cl1}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\overset{cl3}{\bullet}$ et $\overset{cl3}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\overset{cl2}{\bullet}$
$\overset{cl1}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\overset{cl1}{\bullet}$ et $\overset{cl1}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\overset{cl1}{\bullet}$ se réduit en $\overset{cl1}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\overset{cl3}{\bullet}$ et $\overset{cl3}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\overset{cl1}{\bullet}$

Sommaire sur l'autostabilisation

- Définition d'algorithmes indépendants des états initiaux
- Algorithmes très complexes à vérifier en particulier *la terminaison*
- Version autostabilisante de beaucoup d'algorithmes

Terminaison de l'algorithme

- stabilité : un seul processus a la main ou un seul $i \in 0..N \wedge v[i] = v[i{+}_N 1]$.
- propriété 1 : il y a au moins un processus avec une garde franchissable :
  - tous les processus de 1 à N ont la même valeur (hypothèse)
  - nécessairement 0 et N ont la même valeur et 0 a la main.
- propriété 2 : toute configuration légale satisfait la propriété de clôture :
  - les processus de 0 à i-1 ont la même valeur
  - les processus de i à N ont la même valeur
  - $v[i] \neq v[i-1]$
- propriété 3 : à partir de toute configuration illégale, l'anneau atteint une configuration légale (le nombre de processus ayant la main ne peut pas croître et au mieux il peut stagner)
  - $0 \ldots i-1$: non franchissable
  - chaque processus maintient le nombre de processus franchissables : on construit à partir de p, une suite w1 ... wN différent deux à deux
  - cette suite ne peut pas rester indéfiniment stable pr propagation de la valeur et respect de la constnace.

Conclusion

- Algorithmes répartis : problème de l'expression locale de la globalité
- Algorithmes très complexes à vérifier
- Prise en compte de nombreux aspects de l'environnement comme les fautes, les erreurs, ...

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