Autostabilité
Cours Master 2, 2018

Michel Habib
habib@irif.fr
http://www.irif.fr/~habib

19 février 2018

Plan

Modèles de panne dans un système distribué

Protocoles autostabilisants

Le fameux protocole autostabilisant d'exclusion mutuelle de Dijkstra

Synchronisation d'horloges à l'aide de protocoles autostabilisants

Modèles de panne dans un système distribué

Protocoles autostabilisants

Le fameux protocole autostabilisant d'exclusion mutuelle de Dijkstra

Synchronisation d'horloges à l'aide de protocoles autostabilisants

Il y a trois critères pour différencier les pannes d'un système distribué :

1. Suivant la durée :
   permanente ; transitoire ; intermittente.
   Les pannes temporaires ont donné lieu à la notion d'autostabilité.
2. Suivant la nature de la panne :
   Crash : le processeur ne répond plus, dans certains cas on peut avoir quand même accès aux variables du processeur.
   Panne byzantine ou comportement arbitraire (voire malin) ;
   Corruption de mémoire ;
3. Locale ou globale.

- Exemple typique de panne transitoire :
  une machine qui se réinitialise (très fréquent pour certains systèmes buggés). Pendant le temps de la réinitialisation la machine ne répond plus aux messages du réseau.
- Ce qui est en panne (ou attaqué) c'est l'état du processeur, sa mémoire, ou encore l'état de ses canaux d'entrée–sortie.
- Mais nous supposerons que l'état du programme d'un processeur ne peut pas être modifé.

Modèles de panne dans un système distribué

Protocoles autostabilisants

Le fameux protocole autostabilisant d'exclusion mutuelle de Dijkstra

Synchronisation d'horloges à l'aide de protocoles autostabilisants

Définition : Un protocole est autostabilisant, si en cas de panne transitoire, à l'issue de cette panne et s'il n'y en a pas d'autre, le système converge vers un état dans lequel le système fonctionne correctement.
Notion inventée par Dijkstra en 1974 [1], qui proposa un protocole autostabilisant d'exclusion mutuelle sur un anneau de processeurs.

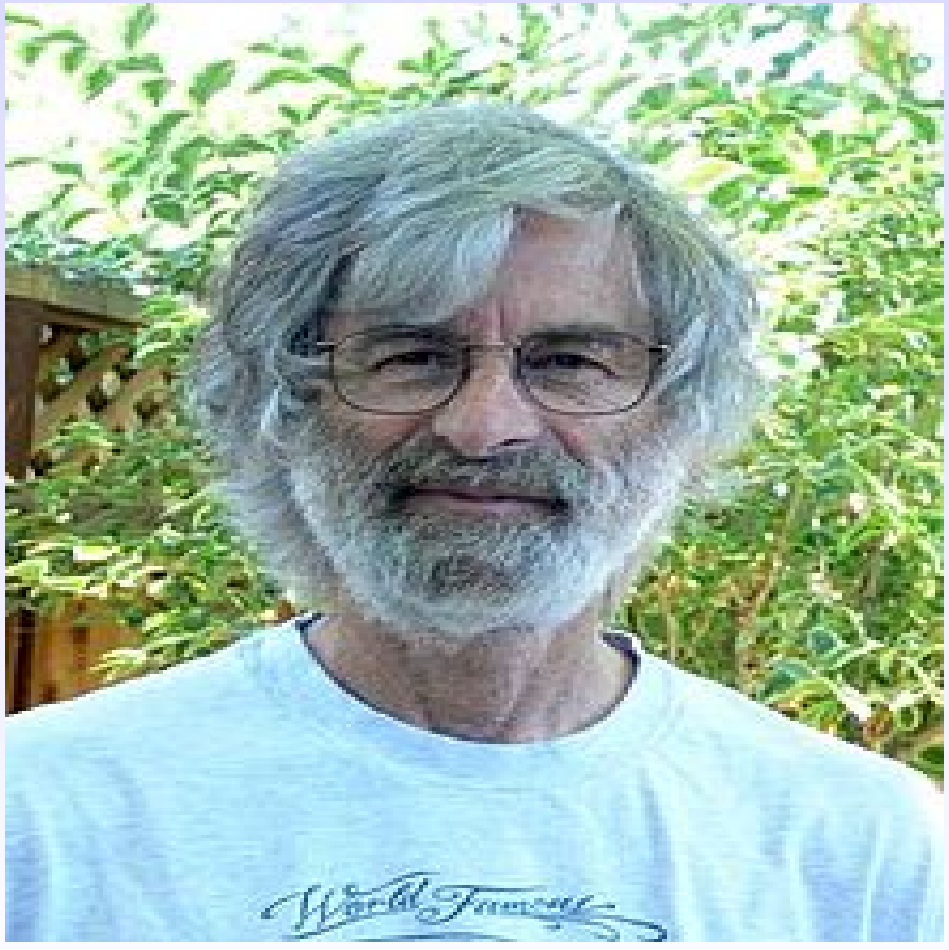
On pourrait généraliser l'autostabilité au cas où subsiste quelques processeurs définitivement en panne.

Edsger W. Dijkstra

Leslie Lamport en 1984 :

- " I regard this as Dijkstra's most brilliant work - at least, his most brilliant published paper. It's almost completely unknown. I regard it to be a milestone on work on fault tolerance "
- Plus loin :
  "I regard self-stabilization to be a very important concept in fault tolerance, and to be a very fertile field for research. "
- Il avait grandement raison !

Leslie Lamport



- On considère un réseau qui est un cycle à n sommets (de 0 à n-1).
- Chaque processeur p_i est dans un état $\sigma_i \in \{0, k-1\}$ avec $k >> n$.
- Le système est synchrone. Les processeurs ont leurs horloges synchronisées.
- Le processeur p_i connaît son état et peut lire l'état de son prédécesseur sur le cycle.

Le programme parallèle :

- p_i avec $i \neq 0$: si $\sigma_i \neq \sigma_{i-1}$ alors $\sigma_i \leftarrow \sigma_{i-1}$
- Pour p_0 : si $\sigma_0 = \sigma_{n-1}$ alors $\sigma_0 \leftarrow \sigma_{n-1} + 1 \bmod k$

Les privilèges

- p_i avec $i \neq 0$ possède le privilège si $\sigma_i \neq \sigma_{i-1}$
- p_0 possède le privilège si $\sigma_0 = \sigma_{n-1}$

Preuve d'autostabilité

D'abord constatons qu'il y a deux situations légales :

1. Si aucun processeur p_i avec $i \neq 0$ ne possède le privilège, alors $\sigma_1 = \sigma_2 \cdots = \sigma_{n-1} = \sigma_0$
   et donc p_0 possède le privilège.
2. Si un et un seul processeur p_i possède le privilège. Alors $\forall j \in [i+1, n-1]$, p_j est dans l'état σ_i. σ_0 est différent σ_i, sinon p_0 aurait le privilège et pour la même raison : $\forall j \in [1, i-1]\ \sigma_j = \sigma_0$.

Une fois dans un état légal de l'un des deux types précédents, après, à chaque pas d'exécution le privilège se décalera d'un sommet à son voisin dans le cycle.
Et le système restera dans un état légal, sauf en cas de nouvelle panne temporaire.

Début de la preuve générale

Cas simple

On oublie pour l'instant le calcul modulo.
Si $\forall i \in [1, n-1]$, $\sigma_i \neq \sigma_0$, alors le processeur p_0 n'aura le privilège que lorsque σ_0 aura fait le tour du cycle.
Après le système sera dans un état légal et le restera, sauf en cas de nouvelle panne temporaire.

Deuxième cas

En au plus n étapes on se ramène à un cas du type précédent

On peut aussi remarquer que si p_i et p_{i+1} avec $i > 0$ ont le même état, alors ils vont se comporter pareillement pendant toute l'exécution et on peut n'en garder qu'un, ce qui permet de faire une preuve par induction.
En outre cela permet de décrire le plus mauvais cas, et d'en déduire que l'algorithme est en $O(n)$.

Nota Bene

Même quand l'anneau est dans une configuration légale il ne le sait pas !
En plus sans coordination centralisée cette connaissance est difficile à mettre en oeuvre.
Cette remarque vaut pour tous les algorithmes autostabilisants.

Exercices

1. Quelle est la valeur minimale de k en fonction de n qui assure l'autostabilité ?
2. Et si on modifiait la règle des p_i par prendre le maximum ?

Exam 2015 : Variations sur l'algorithme d'exclusion mutuelle de Dijkstra

Rappelons le protocole :

- On considère un réseau qui est un cycle à n sommets (de 0 à n-1).
- Chaque processeur p_i est dans un état $\sigma_i \in \{0, k-1\}$ avec $k >> n$.
- Le système est synchrone. Les processeurs ont leurs horloges synchronisées.
- Le processeur p_i connaît son état σ_i et peut lire l'état σ_{i-1} de son prédécesseur sur le cycle.

Le programme parallèle :

- p_i avec $i \neq 0$: si $\sigma_i \neq \sigma_{i-1}$ alors $\sigma_i \leftarrow \sigma_{i-1}$
- Pour p_0 : si $\sigma_0 = \sigma_{n-1}$ alors $\sigma_0 \leftarrow \sigma_{n-1} + 1 \bmod k$

Les privilèges

- p_i avec $i \neq 0$ possède le privilège si $\sigma_i \neq \sigma_{i-1}$
- p_0 possède le privilège si $\sigma_0 = \sigma_{n-1}$

Les questions de cours

1. A quoi peut bien servir un algorithme d'exclusion mutuelle ? Donner des exemples d'application.
2. * Peut-on modifier le protocole afin de savoir si le système est dans un état légal ? Un état légal du système est un état global dans lequel une et une seule machine possède le privilège.

On considère maintenant le protocole Dijkstrabis définit comme suit (on garde les mêmes hypothèses sur la configuration en anneau).

Le programme parallèle :

- p_i avec $i \neq 0$: si $\sigma_i < \sigma_{i-1}$ alors $\sigma_i \leftarrow \sigma_{i-1}$
- Pour p_0 : si $\sigma_0 \leq \sigma_{n-1}$ alors $\sigma_0 \leftarrow \sigma_{n-1} + 1 \bmod k$

Les privilèges

- p_i avec $i \neq 0$ possède le privilège si $\sigma_i < \sigma_{i-1}$
- p_0 possède le privilège si $\sigma_0 \leq \sigma_{n-1}$

Les questions

1. Montrer qu'à chaque étape, il existe toujours au moins un processeur possédant le privilège.
2. * Nous avons donc deux algorithmes Dijkstra et Dijkstrabis qui vérifient cette propriété, pouvez vous généraliser (c'est à dire proposer une famille d'algorithmes qui vérifient cette propriété).
3. Exhiber des situations dans lesquelles le système se comporte bien. C'est à dire, s'il n'y a pas de nouvelle panne transitoire, le privilège tourne sur l'anneau qui passe d'un état légal à un autre état légal.

Les questions suite

1. * Le protocole Dijkstrabis est-il autostable ? Si oui donner une preuve, sinon un contre-exemple.
2. Variations sur le réseau.
   Peut-on adopter l'algorithme de Dijkstra si le réseau est un chemin symétrique $p_1, \ldots p_n$ de processeurs ? Dans ce cas chaque processeur p_i avec $i \neq 1, n$ connaît son état σ_i ainsi que les états de ses voisins sur le chemin σ_{i-1} et σ_{i+1}. Aux deux extrémités du chemin, p_1 connaît σ_1 et σ_2 et p_n connaît σ_n et σ_{n-1}.
3. * Même question lorsque le réseau est un arbre symétrique ?

Applications

- Tout système informatique distribué et autonome devrait être autostabilisant.
- Les algorithmes de routage dans les réseaux, doivent être par nécessité autostabilisants, car il existe de nombreuses pannes transitoires de liens ou de sommets.
- Pour ce faire, dans RIP par exemple, les tables de routage sont totalement recalculées périodiquement, ce qui assure la mise à jour des informations sur le réseau.

On considère un réseau connexe de processeurs synchrones R.

- A chaque étape les processeurs font $+1$ à leur horloges notées $h(P_i)$.
- On suppose que chaque processeur peut connaître à chaque étape les horloges de ses voisins dans R et l'on notera $diam(R)$ le diamètre du réseau R (i.e. la longueur de la plus longue distance entre deux points du réseau).
- Ce qui suit s'inspire d'un article de synthèse [2].

Il y a deux protocoles bien connus Max et Min :

1. A chaque étape, $h(P_i) = max(h(P_i), h(Q)_{Q \text{ voisin de } P_i}) + 1$
2. A chaque étape, $h(P_i) = min(h(P_i), h(Q)_{Q \text{ voisin de } P_i}) + 1$

Théorème

Les protocoles Min et Max sont autostabilisants et leur convergence est en $O(|R|)$.

Preuve

Raisonnons dans le cas du protocole Max.
Propriété fondamentale : Si la valeur de l'horloge d'un processeur à une étape t est égale au maximum des valeurs, alors cette propriété est conservée par la suite si le processeur ne tombe pas en panne temporaire.
Notons A_t l'ensemble des processeurs ayant une horloge maximale à l'étape t.
Si $A_t = R$, les horloges du réseau sont toutes synchronisées à l'étape t. Dans le cas contraire, il existe au moins un $x \notin A$ adjacent à un élément de A_t. Donc $x \in A_{t+1}$ et ainsi l'ensemble grossit strictement d'une étape à l'autre.
Donc en au plus $O(|R|)$ étapes toutes les horloges ont la même valeur.

Cependant il est possible d'améliorer l'analyse de convergence.

Théorème

Les protocoles Min et Max sont autostabilisants et leur convergence est en $O(diam(R))$.

Preuve

Reprenons le raisonnement précédent. Le pire cas est celui dans lequel $|A_0| = 1$. Mais dans ce cas la synchronisation revient à faire une diffusion dans le réseau.
Donc en au plus $diam(R)$ étapes les horloges des processeurs indiquent la même valeur.

N.B. la preuve est valable pour les deux protocoles.
Dans les protocoles précédents les horloges ne sont pas bornées, mais pour certaines applications c'est une condition nécessaire.
Pour synchroniser des horloges de taille bornée, il suffit de compter modulo M, les protocoles deviennent :

1. A chaque étape,
   $h(P_i) = max(h(P_i), h(Q)_{Q \text{ voisin de } P_i}) + 1 \mod M$
2. A chaque étape,
   $h(P_i) = min(h(P_i), h(Q)_{Q \text{ voisin de } P_i}) + 1 \mod M$

Dans ces derniers protocoles, lorsqu'une horloge atteint la valeur $M-1$, à létape suivante l'horloge a pour valeur 0. Dans ces protocoles la propriété fondamentale démontrée précédemment n'est plus vraie.

Cependant nous pouvons obtenir les résultats suivants :

Lemme

Si toutes les horloges du réseau sont inférieures à M-$diam(R)$ alors le protocole converge en $O(diam(R))$.

Preuve

En effet dans ce cas on peut appliquer le résultat du théorème précédent.

Théorème

Si $M \geq (|R| + 1).diam(R)$, le protocole Max modulo est autostabilisant et sa convergence est en $O(M.diam(R))$.

Preuve

Si l'état du réseau ne correspond pas au lemme précédent, il est facile de voir (principe des tiroirs : si l'on met N+1 chaussettes dans n tiroirs, il existe au moins un tiroir qui contient deux chaussettes) qu'il existe au moins deux processeurs P, Q, tels que $h(Q) - h(P) > diam(G)$, et il n'y a pas d'autre horloge dans cet intervalle $[h(P), h(Q)]$.
En effet supposons le contraire : les intervalles entre deux horloges sont tous $\leq diam(G)$ et donc $M \leq (|R| + 1).diam(R)$ d'où la contradiction.
Il suffit alors d'attendre l'état dans lequel $h(Q) = 0$ afin d'appliquer le résultat du lemme précédent. La convergence est donc en $O(M.diam(R))$.

Remarques : Ainsi lorsque M est choisi trop petit, le système peut osciller et ne plus atteindre un état dans lequel toutes les horloges seraient synchronisés.
On démontre un théorème analogue dans le cas du protocole Min modulo.

Exercice : extrait de l'examen 2012

1. Que peut-on dire de l'horloge suivante ?
   À chaque étape :
   $h(P_i) =$
   $Choix(min(h(P_i), h(Q)_{Q \text{ voisin de } P_i}), max(h(P_i), h(Q)_{Q \text{ voisin de } P_i}) + 1.$
   La fonction Choix étant la même pour tous les processeurs.
2. Si l'on prend le même protocole, mais que le résultat de la fonction Choix dépende du processeur. Que peut-il se passer ?


Edsger W. Dijkstra.
Self-stabilizing systems in spite of distributed control.
Commun. ACM, 17(11) :643–644, 1974.


Shlomi Dolev.
Stabilizing in the presence of faults, the digital clock synchronization case.
In *OPODIS*, pages 285–292, 1997.