Systèmes et algorithmique répartis

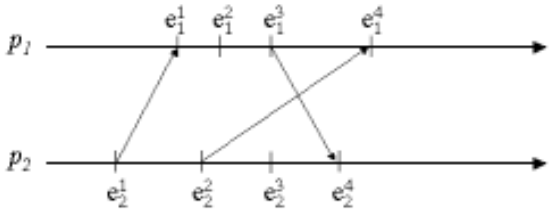
ENSEEIHT/DIMA
1h30, documents autorisés

4 décembre 2018

1 Questions de cours (7 points)

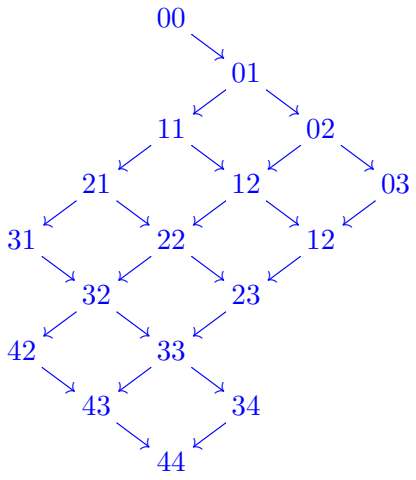
1. (1 point) On considère un système de trois processus muni d'horloges vectorielles. On considère deux événements e et e' respectivement datés par $(2, 1, 5)$ et $(6, 3, 5)$. Quelle est la relation de précédence causale entre e et e' ?
   $e \prec e'$

2. (2 points) On considère le système décrit par le chronogramme ci-dessous :



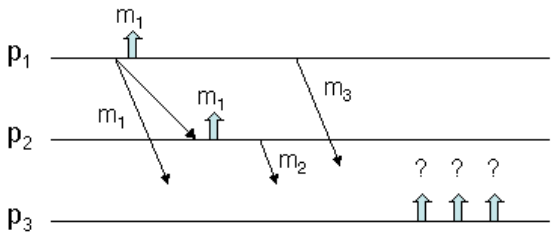
   Représenter graphiquement le treillis des états cohérents de ce système.

   état $xy = (e_1^x, e_2^y)$



3. (1 point) Sur ce même chronogramme, on considère les deux états (e_1^3, e_2^2) et (e_1^2, e_2^3). On suppose qu'une propriété P du système est vraie dans ces deux états. Que peut-on dire de P, du point de vue de l'ensemble des exécutions possibles du système ?
   P est inévitable (Def(P)).

4. (1 point) Même question pour le couple d'états (e_1^2, e_2^1) et (e_1^1, e_2^2) ?
   P est possible, sans plus de certitude (pas d'information pour (e_1^0, e_2^3)).
5. (1 point) Donner un chronogramme de deux processus communiquant par messages, et tels que toute coupure cohérente (sauf l'état initial) comporte au moins un message en transit (c'est-à-dire qui traverse la coupure). Le schéma le plus simple est le meilleur. Indications : utiliser une structure régulière (répétitive) pour chaque processus.
   Chaque processus émet un message avant de recevoir celui de l'autre.
   Chaque message (sauf le tout premier) croise deux messages venus de l'autre site.
   TODO : dessin à faire.
6. (1 point) On considère le système représenté ci-dessous, utilisant à la fois la communication point à point et la diffusion de messages. Les flèches verticales indiquent la délivrance des messages. On demande de remplacer les ? par des numéros de messages (m_1, m_2, m_3) de manière que la délivrance des messages soit FIFO, mais non causale.



   - *m_1 puis m_3 puis m_2 : m_1 avant m_3 pour respecter la délivrance FIFO, m_3 avant m_2 pour que ça ne respecte pas la délivrance causale.*
   - *m_2 puis m_1 puis m_3 : m_1 avant m_3 pour respecter la délivrance FIFO, m_2 avant m_1 pour que ça ne respecte pas la délivrance causale.*

2 Diffusion fiable dans un système asynchrone (3 points)

On suppose un système asynchrone avec défaillances d'arrêt (fail stop) sans reprise (la défaillance est définitive). Un processus est dit correct s'il n'est pas défaillant. On suppose que tout processus correct peut toujours communiquer avec tout autre processus correct (les défaillances ne partitionnent pas l'ensemble des processus corrects).

On veut réaliser un protocole de diffusion fiable uniforme, défini par les propriétés suivantes : si un processus correct diffuse un message m, ce processus délivre m ; si un processus (correct ou incorrect) délivre un message m, tous les processus corrects délivrent ce message m ; si un processus correct délivre m, alors il le délivre une seule fois, et m a été diffusé par un processus.

On propose l'algorithme suivant pour la diffusion fiable :

- Exécution de *broadcast(m)* par processus p (diffusion de m) :

  `Marquer` m `avec (émetteur de` m`, numéro de séquence) // identifie` m
  `Exécuter` *send*(m) `vers tous les voisins (y compris` p`)`

- Exécution de *receive(m)* par processus p (réception de m) :

```
Lors de la première exécution de receive(m) :
Exécuter deliver(m) (au processus p)
Exécuter send(m) vers tous les voisins
```

Les primitives *send* et *receive* sont les opérations élémentaires d'envoi et réception de message point à point, supposées fiables (un message envoyé parvient à destination si l'émetteur et le récepteur sont corrects). Les voisins d'un processus p sont les processus q tels qu'il existe un canal direct de communication de p vers q.

Questions

1. Montrer que l'algorithme ci-dessus est incorrect, en donnant un scénario qui viole les spécifications.

   Le noœud initial envoie à tous ses voisins. Tous ceux-ci (y compris le nœud initial) sont incorrects : ils reçoivent le message, le délivrent puis s'arrêtent avant de l'envoyer au reste du monde. Le message a été délivré à certains mais pas à tous ⇒ non uniforme.

   Note : l'hypothèse du maintien de la connexité concerne les nœuds corrects. Un cas comme le graphe $p_0 \leftrightarrow p \leftrightarrow$ reste), avec p_0 site correct initial, et p incorrect qui délivre le message puis s'arrête avant de l'envoyer au reste, n'est pas un contre-exemple valide : la connexité entre p_0 et le reste n'est pas maintenue (c'est pour cela qu'il faut aussi p_0 incorrect).

2. Corriger l'algorithme.

   Inverser les lignes délivrer / envoyer aux voisins. L'algorithme n'est correct que si le graphe reste connexe (pas de partition, ce qui est en hypothèse)

3. Quel est l'intérêt pratique de considérer un protocole de diffusion uniforme (c'est-à-dire garantissant qu'un processus incorrect ne peut pas délivrer un message sans que les corrects ne le délivre aussi) ?

   - *Présence d'effet de bord sur le reste du monde : tous les processus peuvent le faire ou aucun.*
   - *Redémarrage plus facile : on sait précisément les messages reçus par un processus avant qu'il ne crashe (= tous). Avec la diffusion fiable non uniforme, un processus qui va plus tard s'arrêter peut ne pas délivrer des messages bien avant son crash (vu qu'il est finalement défaillant, il ne compte pas !). Un processus silencieux (= qui ne diffuse pas d'informations aux autres) et finalement défaillant peut avoir cessé de délivrer des messages sans que qu'aucun processus ne le sache : son état au moment du crash est donc inconnu.*

3 Détecteur de défaillance (3 points)

On suppose toujours le même système asynchrone avec défaillance d'arrêt. On rappelle qu'un détecteur de défaillances construit et maintient à jour, pour chaque processus, une liste de processus suspectés d'être défaillant par ce processus. On rappelle que la classe S de détecteurs de défaillances est caractérisée par les propriétés ⟨complétude forte, exactitude faible⟩ et la classe W par les propriétés ⟨complétude faible, exactitude faible⟩.

Questions

1. En supposant disponibles un détecteur de défaillances de la classe W ainsi qu'une primitive de diffusion fiable *broadcast(m)*, telle que spécifiée au paragraphe 2, expliquer comment construire un détecteur de la classe S et donner l'algorithme correspondant.
   Réponse dans le poly, chapitre « consenus », transparent 33 (Complétude).

4 Consensus dans un système synchrone (7 points)

On considère maintenant un système synchrone ; la durée de transmission d'un message est bornée par une constante connue Δ. Un algorithme est alors décrit comme une suite de tours synchrones. On étudie l'algorithme de consensus tolérant au plus f défaillances d'arrêt et avec décision anticipée (*early consensus*) basé sur l'égalité des messages reçus à deux tours consécutifs :

La formulation de l'algorithme est fragile : la décision anticipée d'un processus est s'il a reçu le même ensemble de messages ⟨valeur,émetteur⟩, ou de manière équivalente le même multi-ensemble (multiset) *de valeurs. Il est erroné de faire de la décision anticipée avec le même* ensemble *de valeurs deux tours consécutifs : l'ensemble pourrait être identique mais obtenu avec moins de messages (même valeur proposée par plusieurs processus, certains ayant failli) ⇒ défaillance non détectée !*

```
Processus P_i(v_i),  0 ≤ i < n
local x, early, received, recpred
on start :
    x ← v_i
    early ← false
    received ← ∅
    recpred ← ⊥
on réception(v) :
    received ← received ∪ {{v}}
on round k, 0 ≤ k ≤ f ∧ ¬early:     // à chaque tour avant décision
    x ← min(received ∪ {{x}})
    if received = recpred then
        early ← true                 // (*)
    fi
    recpred ← received
    received ← ∅
    envoyer(x) à tous les autres
on early = true:
    décider x  // et se terminer
on round (f + 1):
    décider x  // et se terminer
```

Questions

1. Montrer que cet algorithme a la propriété d'accord (la valeur décidée est la même pour tous les processus corrects).
   Lorsqu'un processus décide par anticipation, c'est qu'il a reçu les mêmes valeurs deux tours consécutifs. Donc (pour lui), il n'y a pas de défaillance supplémentaire au second de ces tours et il sait que les valeurs reçues au premier de ces tours contenaient l'intégralité des valeurs en circulation [noter qu'un processus a pu s'arrêter après lui avoir envoyé sa valeur et avant

de l'envoyer aux autres : pour ces autres processus, il n'y a pas l'égalité des valeurs reçues, tous ne font pas simultanément une décision anticipée]. Au second tour, il garde le même min, qu'il avait diffusé au tour précédent ⇒ tous les autres processus corrects l'ont reçu et le choisiront ⇒ décision identique.

S'il n'y a pas de décision anticipée, il existe un tour sans défaillance. Suite à ce tour, tous les processus corrects prennent la même valeur et n'en changent plus. Ils envoient donc les mêmes messages à chaque tour et la valeur décidée est le min de celles qui ont été vues par les processus corrects.

2. Monter qu'il a la propriété d'intégrité (tout processus décide au plus une fois).

   Unique point de décision, soit au round $f+1$, soit en cas de décision anticipée.

3. Monter qu'il a la propriété de validité (la valeur décidée est l'une des valeurs proposées).

   Seules les valeurs proposées sont envoyées. La valeur de décision est un min des valeurs reçues, donc un min d'un sous-ensemble des valeurs proposées. Et le min d'un ensemble est dans l'ensemble.

4. Monter qu'il a la propriété de terminaison (tout processus correct décide au bout d'un temps fini).

   Décision au plus tard au tour $f+1$.

5. Donner un exemple avec $f = 2$ où un processus décide en strictement moins de $f+1$ tours.

   Initialement (tour 0), 2 processus s'arrêtent avant même de diffuser, tous les autres diffusent leur valeur. Au tour 1, tous les processus sont corrects, ont calculé le même min et l'envoient à tous. Au tour 2, tous les processus reçoivent de nouveau les mêmes messages (tous contenant le min) et concluent en avance.

6. Quand un processus détecte qu'il peut décider en avance (ligne `(*)`), expliquer pourquoi il est important qu'il ne se termine pas immédiatement (en faisant une action `decider`) mais qu'il réalise encore un envoi à tous les autres.

   Cela permet aux autres processus de faire une décision en avance. Soit un processus p faisant une décision anticipée. On peut être dans le cas où p avait reçu un message d'un processus défaillant qui ne l'avait pas envoyé aux autres. p a pu décider mais pas les autres. En envoyant encore une fois sa valeur, p garantit que les autres reçoivent les mêmes messages au tour suivant (s'il n'y a pas de nouvelles défaillances), et ils peuvent à leur tour décider par anticipation.

   Par contre, la ligne (*) n'est pas nécessaire pour l'accord : vu que le processus qui fait une décision anticipée reçoit exactement les mêmes valeurs, sa valeur candidate au tour précédent est la même que sa valeur candidate à ce tour (celle qui va être la valeur décidée). Cette valeur a donc déjà été diffusée aux autres, et tous les corrects la connaissent.

7. Montrer que le pire cas reste $f+1$ tours.

   Cas où, à chaque tour, un unique processus possède le min, le transmet à un seul autre puis s'arrête (cf cours « consensus », transparent 21).