M1if12 Algorithmique Distribuée

Contrôle 3 - 2020-2021

Durée : 1 heure

Aucun document autorisé - Calculatrice (non sur téléphone) et règle autorisées

L'énoncé avec les réponses aux questions sera rendu dans une copie d'examen. Le numéro d'anonymat de la copie d'examen sera reporté sur l'énoncé dans le cadre ci-dessous.

Numéro d'anonymat (à reporter de la copie d'examen) : **Il ne s'agit pas de votre numéro étudiant !**

Barême donné à titre indicatif :

Exercice 1	Exercice 2	Exercice 3	Exercice 4
4 pts	6 pts	6 pts	4 pts

1 Exercice - Synchronisation en temps avec le système GPS

Considérons un système GPS simplifié. Considérons un récepteur GPS R ayant pour coordonnées (x, y, z) dans un système de coordonnées fixé. Considérons 3 satellites GPS : S_1 de coordonnées (x_1, y_1, z_1), S_2 de coordonnées (x_2, y_2, z_2) et S_3 de coordonnées (x_3, y_3, z_3). On supposera que les signaux GPS se déplacent à une vitesse de $c = 3.10^8$ m/s.

Les valeurs qui seront manipulées par la suite ne correspondent à aucune réalité.

Questions :

1. Si un signal GPS met 70 μs pour aller d'un satellite GPS au récepteur R, quelle est la distance séparant ce satellite de R ?

   distance = temps $\times$ c = $70.10^{-6} \times 3.10^8 = 210.10^2 = 21$km.

2. Supposons que les 3 satellites GPS émettent leur signal GPS en même temps à l'instant t et que R reçoit, dans le référentiel temporel GPS, le signal de $S1$ à t_1, le signal de $S2$ à t_2 et le signal de $S3$ à t_3. Écrire, pour chaque satellite S_i, l'équation qui relie les différents instants (t et t_i) avec les cordonnées de R et de S_i.

Pour chaque satellite i, on a la relation suivante : $t_i - t = \frac{\sqrt{(x_i-x)^2+(y_i-y)^2+(z_i-z)^2}}{c}$

3. Il est possible que l'horloge de R ne soit pas bien synchronisée sur le référentiel de temps GPS. Sachant que le signal provenant de S_1 arrive en R 1μs avant celui de S_2 qui lui-même arrive en R 1μs avant celui de S_3. Écrire les équations dont R aura besoin pour déterminer ses coordonnées permettant de contourner le problème de synchronisation de R. Il n'est pas demandé de résoudre ces équations.

$$t_1 - t_2 = \frac{\sqrt{(x_1-x)^2+(y_1-y)^2+(z_1-z)^2}}{c} - \frac{\sqrt{(x_2-x)^2+(y_2-y)^2+(z_2-z)^2}}{c} = -10^{-6}$$

$$t_1 - t_3 = \frac{\sqrt{(x_1-x)^2+(y_1-y)^2+(z_1-z)^2}}{c} - \frac{\sqrt{(x_3-x)^2+(y_3-y)^2+(z_3-z)^2}}{c} = -2.10^{-6}$$

$$t_2 - t_3 = \frac{\sqrt{(x_2-x)^2+(y_2-y)^2+(z_2-z)^2}}{c} - \frac{\sqrt{(x_3-x)^2+(y_3-y)^2+(z_3-z)^2}}{c} = -10^{-6}$$

4. Une fois que R a déterminé ses coordonnées, expliquez comment il synchronise son horloge sur le référentiel de temps GPS

Un message GPS contient les coordonnées du satellite qui l'émet ainsi que le temps auquel le message a été envoyé (t). Quand R a déterminé ses coordonnées, il peut déterminer sa distance entre le satellite et lui-même et donc calculer le temps pris par le message entre le satellite et lui-même. Il synchronise alors son horloge en prenant comme temps t + temps pris par le message.

2 Exercice - Temps logique

Considérons un programme distribué s'exécutant sur 3 processus dont l'exécution est donnée ci-dessous. Dans cette exécution, un rond sans flèche correspond à une action interne, un rond au départ d'une flèche correspond à l'envoi d'un message tandis qu'un rond au bout d'une flèche correspond à la réception d'un message.

1. Compléter l'exécution du programme avec les horloges vectorielles pour chaque processus.

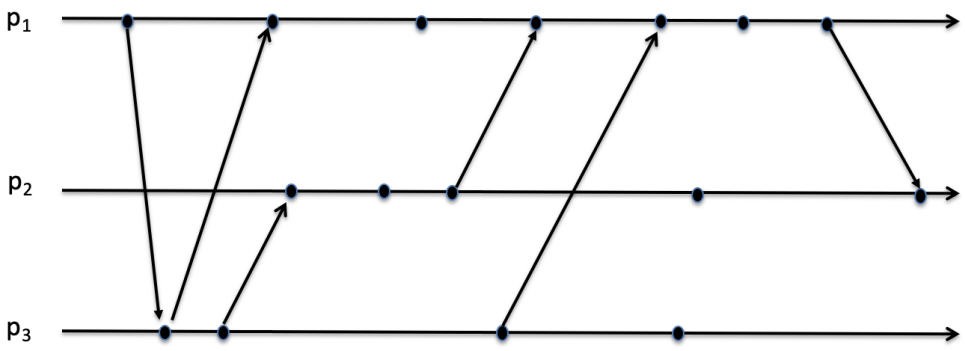

FIGURE 1 – Exécution d'un programme distribué

2. Montrer que les horloges vectorielles vérifient la propriété de cohérence d'horloges

Considérons deux actions consécutives sur un même processus, a_i et a_{i+1}. Si a_{i+1} est une action interne, alors l'horloge interne est incrémentée de 1 et les autres composantes de l'horloge sont identiques à celles de a_i. Donc $C(a_i) \leq C(a_{i+1})$. On peut faire la même observation si a_{i+1} correspond à l'envoi d'un message.
Si a_{i+1} correspond à la réception d'un message, le processus incrémente de 1 son horloge interne et met à jour son horloge correspondant au processus qui a envoyé le message (en prenant le maximum entre l'horloge reçue et son horloge). Avec ses opérations, on a encore $C(a_i) \leq C(a_{i+1})$.
Considérons maintenant que a_i précède a_{i+1} et que les 2 actions sont sur 2 processus différents. a_i correspond à l'envoi d'un message et a_{i+1} correspond à la réception de ce message. À la réception de ce message, le processus incrémente de 1 son horloge locale (qui est forcément au moins supérieure ou égale à celle reçue dans le message) et prend le maximum pour toutes les autres horloges. Donc on a $C(a_i) \leq C(a_{i+1})$.
On peut décomposer tout cas de figure correspondant à une action qui précède l'autre, avec les schémas élémentaires précédents.

3. Qu'est-ce-que la propriété de cohérence forte ? Est-ce que les horloges vectorielles vérifient cette propriété ? Il ne vous est pas demandé de le démontrer.

Oui les horloges vectorielles vérifient la propriété de cohérence forte qui correspond à une équivalence entre une action qui précède une autre et la relation d'ordre entre leur horloge.

4. Que peut-on dire des horloges vectorielles de la 3e action du processus p_1 et de la 2e action du processus p_2 ?

Elles sont incomparables.

5. Supposons que l'unique message envoyé du processus p_2 au processus p_1 soit perdu et que p_1 ne le reçoit pas (on supposera que p_1 ne tombe pas en panne pendant l'exécution du programme). Si les protocoles de la pile TCP/IP utilisés pour envoyer ce message n'acquitte pas la bonne réception du paquet, est-ce que p_2 peut néanmoins se rendre compte, plus tard dans le programme, que p_1 n'a pas reçu son message ? Vous expliquerez votre réponse.

Oui p_2 peut s'en rendre compte grâce à l'horloge que p_1 lui envoie. En effet, p_1 devrait au moins avoir l'horloge locale de p_2 quand il lui avait envoyé son message (3). Or p_1 indique que sa connaissance de l'horloge de p_2 est 0. Donc p_2 peut se dire que le message n'a pas été reçu.

3 Exercice - Algorithme d'Itai-Rodeh

L'algorithme d'Itai-Rodeh, vu en cours et en TP, a pour but d'élire un leader dans un anneau unidirectionnel anonyme (c'est-à-dire un anneau où les identifiants ne sont pas uniques). On considère que les processus impliqués dans cet algorithme ne tombe jamais en panne.

1. Quelles sont les hypothèses de cet algorithme ?

i) l'anneau est unidirectionnel et anonyme ; ii) le nombre de nœuds dans l'anneau (n) est connu par tous les nœuds ; iii) les communications sont FIFO.

2. Expliquer les principales étapes de cet algorithme.

L'algorithme fonctionne en phase. Les nœuds actifs choisissent in nombre aléatoire dans un intervalle plus grand que le nombre de nœuds et se déclarent leader. Quand un nœud reçoit un message d'un nœud se déclarant leader, il peut : i) retransmettre le message s'il était inactif ; ii) retransmettre le message s'il est actif et que sa valeur aléatoire choisie est plus petite ainsi que sa phase ; ii) initier un message de non unicité s'il a choisi la même valeur que celle reçue dans les même phase. À chaque fois que le message est transmis est compteur est incrémenté de 1.
Quand un nœud reçoit un message avec un TTL de n, si le message indique que le leader est unique il se déclare leader. Si le message indique qu'il n'est pas unique alors il passe à la phase suivante et tire un nouveau nombre aléatoire.

3. Comment assurer que cet algorithme trouve un leader avec très forte probabilité ?

Il faut choisir la valeur aléatoire dans un grand intervalle. Plus l'intervalle est grand, plus la proba que deux nœuds tirent la même valeur est faible.

4. Quel est le nombre minimal de messages envoyés avec cet algorithme pour trouver le leader et informer tous les nœuds de l'anneau de l'identifiant du leader ? Vous expliquerez votre réponse.

   Le meilleur cas correspond à un seul noeud qui se déclare leader. Dans ce cas, n messages sont envoyés pour que ce nœud considère qu'il est bien leader. Puis n autres messages sont envoyés pour informer tous les autres nœuds.

5. Est-ce qu'avec cet algorithme, un nœud n'ayant pas initialement (au tout début de l'algorithme) le plus petit ID ou le plus grand ID dans l'anneau peut être élu leader ? Vous expliquerez votre réponse.

   Oui c'est possible car le nœud qui sera élu correspond qui aura choisi la plus grande valeur (unique). Et ce nœud peut avoir un ID qui n'est n'y le plus grand, n'y le plus petit.

6. Si un leader est trouvé, est-il possible de rendre l'anneau non anonyme ? Si oui, donnez un algorithme qui permet de rendre l'anneau non anonyme. Vous décrirez votre algorithme sous forme algorithmique.

Oui. Il suffit que le leader démarre la procédure en prenant par ex. l'identifiant 1, puis envoie l'ID 2 au nœud suivant qui envoie l'ID3 au nœud suivant, etc.

int myID ;
Si (leader) alors { myID := 1 envoyer message label(myID+1) ; }
Sinon { attendre message label(val) ; myID := val ; envoyer message label(myID+1) ;
}

7. Supposons que l'anneau ait un processus byzantin. Comment ce processus pourrait se comporter pour qu'un leader ne soit jamais trouvé avec l'algorithme d'Itai-Rodeh ? Expliquez votre réponse.

À chaque fois que le processus byzantin reçoit un message d'un nœud se déclarant leader, il indique qu'il a choisi le même identifiant et donc qu'il n'y a pas unicité. Comme tout message doit passer par lui, on ne peut jamais trouver de leader.

4 Exercice - Algorithmes de consensus

Soit un système asynchrone constitué de n processus, dont f peuvent tomber en panne (pannes franches). Le système est muni d'un détecteur de fautes P. On considère l'algorithme de consensus avec détecteur P en $f + 1$ rondes (algorithme vu en TD).

1. Décrire l'algorithme.

Chaque processus stocke en mémoire un vecteur contenant la valeur proposée par chaque processus. Au début, chacun ne connaît que sa propre valeur. À chaque ronde, chaque processus diffuse son vecteur à tous les autres, puis attend de recevoir les vecteurs de tous les processus qu'il ne suspecte pas. Après f + 1 rondes, chaque processus décide la première valeur non vide de son vecteur.

2. Pourquoi l'algorithme utilise-t-il $f+1$ rondes ? Que se passerait-il si les processus décidaient après une seule ronde ? Donner un exemple d'exécution problématique.

Un processus plante au milieu de son broadcast, sa valeur n'est transmise qu'à un seul autre processus. Il se peut alors que les processus restants décident des valeurs différentes. Si à chaque ronde le seul processus à avoir reçu la valeur du fautif tombe en panne à son tour après avoir transmis cette valeur à un seul autre processus, alors il faut $f + 1$ rondes pour garantir la cohérence.

3. On utilise maintenant un détecteur $\diamond S$ au lieu du détecteur P. L'algorithme est-il toujours capable de résoudre le consensus ? Justifier.

Non car $\diamond S$ peut causer des fausses suspicions, et permettre à un processus de décider trop tôt. Par exemple : un processus soupçonne tout le monde, passe les $f + 1$ rondes et décide immédiatement, indépendamment du reste du système (violation de la cohérence). Autre réponse possible : non car résoudre un consensus avec $\diamond S$ nécessite une majorité de corrects, hypothèse manquante.

On souhaite maintenant utiliser l'algorithme du coordinateur tournant pour résoudre le consensus avec $\diamond S$.

4. Peut-on prédire le nombre de rondes nécessaires avant que l'algorithme ne se termine ? Expliquer.

Non, car $\diamond S$ ne fournit pas de garantie sur le temps qu'il faut avant qu'un processus ne soit plus suspecté. On sait qu'à terme l'algorithme parviendra à progresser, mais on il est impossible de prévoir le nombre de rondes nécessaires.

5. Combien de pannes l'algorithme du coordinateur tournant peut-il tolérer ? Quel problème se pose si une panne de trop a lieu ?

Il faut une majorité de corrects (et donc, une minorité de pannes). Dans l'algo, le coordinateur attend de recevoir des réponses d'une majorité de processus. Si une majorité de processus sont en panne, alors le coordinateur attendra pour toujours.