Systèmes et algorithmes répartis

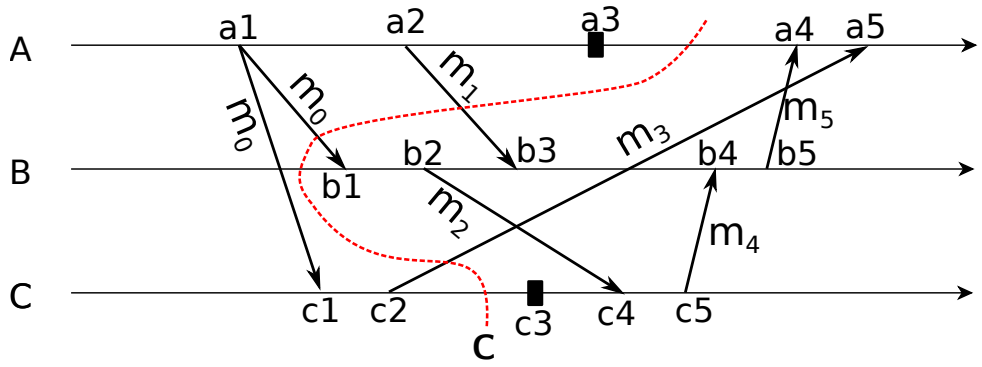
ENSEEIHT/3SN
1h30, documents autorisés

6 décembre 2022

1 point par question, sauf indiqué en marge. Total : 21 points.

1 Questions de cours (6 points)

On considère les échanges de messages entre 3 sites A, B, C représentés par le chronogramme suivant :



Questions

1. Quel est le passé causal de c_4 ?
   $\{c_3, c_2, c_1, b_2, b_1, a_1\}$
2. La coupure C est-elle cohérente ? Justifier la réponse.
   Oui, pas de "trou" sur un site, et pas de réception sans son émission.
3. Si la coupure C est cohérente, y inclure un unique événement du calcul la rendant incohérente ; si elle est incohérente, en exclure un unique événement du calcul la rendant cohérente.
   Par exemple, ajouter a_4.
4. Déterminer la valeur de l'horloge de Lamport de b_4. Justifier cette valeur soit par raisonnement, soit en calculant les horloges des événements le précédant.
   $(B, 7)$
5. Déterminer la valeur de l'horloge vectorielle de b_4. Justifier cette valeur soit par raisonnement, soit en calculant les horloges des événements le précédant.
   $(2, 4, 5)$
6. Ce chronographe vérifie-t-il la délivrance causale pour tous les messages ?
   non, m_5 a m_3 dans son histoire causale.

2 Consensus avec crashs initiaux (10 points)

On considère un système composé de N processus (N est fixe et connu des processus) dans lequel au plus $f < N$ processus sont défaillants initialement : ils ne démarrent pas (f inconnu des processus). Il n'y a ensuite aucune défaillance et le réseau est fiable. Chaque processus P_i propose une valeur v_i et l'on va étudier le problème du consensus.

1. Expliquer pourquoi, dans ce cadre de crashs initiaux, tout algorithme de consensus résout aussi le consensus uniforme.

   Les processus initialement défaillants ne décident jamais. Tous les processus qui décident sont corrects.

2. On suppose que le système est synchrone avec δ, le délai maximal de transfert d'un message. Proposer un algorithme de consensus en justifiant pourquoi il est correct.

   Tout le monde envoie sa valeur. Chacun attend δ et décide le min. Après δ, tous les processus ont reçu les mêmes messages $\Rightarrow$ même décision.

3. On considère pour la suite que le système est asynchrone. Expliquer pourquoi, même avec les défaillances limitées que sont les crashs initiaux, le problème du consensus reste impossible à résoudre en asynchrone.

   On ne sait pas combien de processus sont défaillants. Rien reçu d'un processus ne permet pas de savoir s'il est défaillant ou lent.

4. On dispose d'un détecteur de défaillance P (complétude forte, exactitude forte) et on propose l'algorithme de consensus suivant. Au démarrage, tous les processus (non crashés) envoient leur valeur v_i à tous. Ensuite, chaque processus attend un message de tous les processus non suspectés par P. Il calcule alors le mininimum des valeurs proposées. Expliquer pourquoi cet algorithme est correct en détaillant les quatre propriétés du consensus. (2 pts)

   Complétude forte : il existe un moment où tous les crashés sont suspectés → on cesse d'attendre leur message.
   Exactitude forte : aucun correct n'a été suspecté à tort → on a attendu leur message.
   Tous les processus corrects attendent donc un message du même ensemble de processus et construisent le même ensemble de valeurs.
   Accord : même ensemble de messages → même min.
   Intégrité : pas de boucle, une seule décision.
   Validité : on prend le min des v_i reçus, qui est une des valeurs envoyées.
   Terminaison : on recevra finalement un message de tous les corrects car P est exact.

5. On considère le même algorithme avec le détecteur de défaillance $\Diamond P$ (complétude forte, exactitude finalement forte). Expliquer si l'algorithme est correct dans ce cas.

   Incorrect, car on peut suspecter à tort un processus correct, dont on n'attend alors pas le message. Deux processus peuvent décider sans avoir le même ensemble de messages, on n'a plus l'accord.

On considère maintenant que f est connu avec $f < N/2$ et on va utiliser le détecteur de défaillance $\Diamond\Omega$. Quand on l'interroge, le détecteur $\Diamond\Omega$ renvoie un processus correct, mais pendant un certain temps, il ne fait pas nécessairement la même réponse à tous ; il finit par se stabiliser (en temps inconnu) pour envoyer alors le même processus correct à tous les processus. On étudie l'algorithme suivant :

Chaque processus P_i interroge $\Diamond\Omega$, et chaque fois que le détecteur de défaillance répond un processus différent p, P_i envoie sa proposition v_i à p. Quand un processus a reçu $N - f$ propositions, il choisit une valeur arbitraire parmi celles-ci et il envoie un message de décision à tous. Quand un processus reçoit un message de décision, il accepte cette décision.

```
Processus Pi
  constant vi        -- valeur proposée
```

```
  variable pc_i = ⊥   -- résultat du dernier appel à ◇Ω
  variable S_i        -- multi-ensemble des valeurs reçues
on pas encore décidé :  -- de temps en temps
    newp ← ◇Ω
    if (newp ≠ pc_i) then
        pc_i ← newp
        send proposition(v_i) to pc_i
    endif
on reception proposition(v) :
    S_i = S_i ⊎ {v}
on Cardinality(S_i) ≥ N − f :
    v ← choisir une valeur dans S_i
    send decision(v) to all process
on reception decision(v) :
    décider v
```

6. Cet algorithme vérifie-t-il l'*accord* ?

   Non : le détecteur de défaillance fournit d'abord P_1 à une majorité de processus ($> N - f$). Ceux-ci envoie leur proposition à P_1 qui choisit une valeur et envoie la valeur décidée à tous. Avant que la réponse n'arrive, le détecteur fournit P_2 à une autre majorité de processus. Ils envoient leur proposition, P_2 choisit une valeur et l'envoie à tous. Certains vont recevoir la valeur de P_1 puis celle de P_2, d'autres en ordre inverse. Ils n'ont pas la même valeur de décision.

7. Cet algorithme vérifie-t-il l'*intégrité* ?

   Non, plusieurs messages de décision possibles.

8. Cet algorithme vérifie-t-il la *validité* ?

   Oui, la valeur choisie fait partie des valeurs proposées.

9. Cet algorithme vérifie-t-il la *terminaison* ?

   Oui, il y aura nécessairement une majorité au pire quand $\diamond\Omega$ se stabilise sur le même processus.

3 Protocole de cohérence causale (5 points)

On considère un système réparti dont les sites sont fiables et reliés par un réseau maillé, fiable, asynchrone. Le nombre N de sites est fixe. On souhaite réaliser un protocole de cohérence causale léger pour une donnée partagée X, dupliquée sur chacun des N sites. Chaque site s dispose donc d'une copie locale de X, notée X_s. Les opérations possibles sur la donnée X pour le site s sont :

- $\mathsf{X.Read}_s()$ qui retourne la valeur de X déterminée par le protocole de cohérence ;
- $\mathsf{X.Write}_s(\mathsf{v})$ qui fournit une valeur v à écrire dans X.

On suppose que l'on dispose uniquement d'un service de communication point à point, fournissant deux opérations :

- émettre(msg), qui permet l'envoi du message msg à l'ensemble des N sites ;
- une opération de réception de message, qui peut prendre deux formes :
  - une forme synchrone : recevoir(msg) bloque le site appelant l'opération jusqu'à la réception du message msg ;
  - une forme asynchrone : un traitant sur_réception(msg) peut être défini par le site récepteur ; ce traitant est alors appelé par le service de communication quand le message msg est réceptionné sur le site récepteur.

  Pour ces opérations, vous pouvez considérer si vous en avez besoin :
  - que l'émetteur et le récepteur du message sont connus et peuvent être désignés à votre convenance ;
  - que le message est une liste de valeurs du type de votre convenance.

Indication : avec ce service de communication, l'essentiel de l'implantation de l'opération $\mathsf{X.Write}_s(\mathsf{v})$ sur le site s pourra consister à envoyer un message comportant v à chacun des autres sites. Bien entendu, cette implantation comportera d'autres opérations et les messages émis pourront comporter des données supplémentaires ou différentes de v.

Questions

On rappelle que le protocole de datation par horloges de Lamport possède la propriété de validité faible, c'est-à-dire qu'il assure que pour tous événements distincts e et e', si e précède causalement e', alors $HL(e) < HL(e')$, où $HL(x)$ est la date locale associée à l'événement x par le protocole d'horloges de Lamport.

10. Formuler la contraposée de la propriété de vivacité faible. (0 pt)
11. On va utiliser cette formulation pour implémenter un protocole de cohérence causale léger pour l'objet partagé X.
    (a) Lorsqu'un site reçoit une mise à jour précédant causalement sa dernière écriture sur la copie locale de X, doit-il ignorer cette mise à jour, et considérer qu'elle a été écrasée par la dernière écriture sur sa copie locale de X, ou doit-il nécessairement répercuter cette écriture sur sa copie locale, ou encore a-t-il le choix entre ignorer ou répercuter cette mise à jour ? Justifiez votre réponse. (0,5 pt)
    (b) Même question, dans le cas un site reçoit une mise à jour causalement indépendante de sa dernière écriture sur la copie locale de X. (0,5 pt)
    (c) Même question, dans le cas un site reçoit une mise à jour succédant causalement à sa dernière écriture sur la copie locale de X. (0,5 pt)

    Indication pour les questions (a), (b), (c) : il suffit que le protocole garantisse que l'ordre des mises à jour est conforme à l'ordre causal.

(d) Définir les messages échangés et fournir les algorithmes : (1,5 pt)

- de l'opération $\mathsf{X..Read}_s()$;
- de l'opération $\mathsf{X.Write}_s(-)$;
- des traitants éventuellement associés aux réceptions de messages.

12. Normalement, pour ce protocole, et sous les hypothèses choisies au départ, les opérations $\mathsf{x..Read}_s()$ **et** $\mathsf{x.Write}_s(-)$ devraient être non bloquantes, c'est à dire qu'elles devraient fournir un résultat directement, sans nécessiter de synchronisation (d'attente) par rapport aux autres sites. Expliquez pourquoi. (1 pt)
Note : la concision et la précision seront particulièrement prises en compte dans l'évaluation de cette réponse.

13. Ce protocole est qualifié de léger en comparaison à un protocole plus classique qui consisterait à :

- implanter un service de diffusion causale à partir du service de communication point à point, comme présenté en cours ;
- implanter l'opération $\mathsf{X.Write}_s(\mathsf{v})$ par une diffusion causale, la délivrance du message diffusé se traduisant par la mise à jour de la copie locale de X avec la valeur v reçue ;
- implanter $\mathsf{X..Read}_s()$ par une lecture de la copie locale.

(a) Expliquez en quoi ce protocole est plus léger que le protocole classique, autrement dit quel est le gain (significatif) en termes d'opérations exécutées, ou de degré de parallélisme, ou de nombre de messages échangés, ou encore de volume de données échangées. (0,5 pt)

(b) Expliquez quels sont les désavantages de ce protocole par rapport au protocole classique, dans le cas général, ou sous quelles hypothèses ce protocole est plus avantageux que le protocole classique. (0,5 pt)