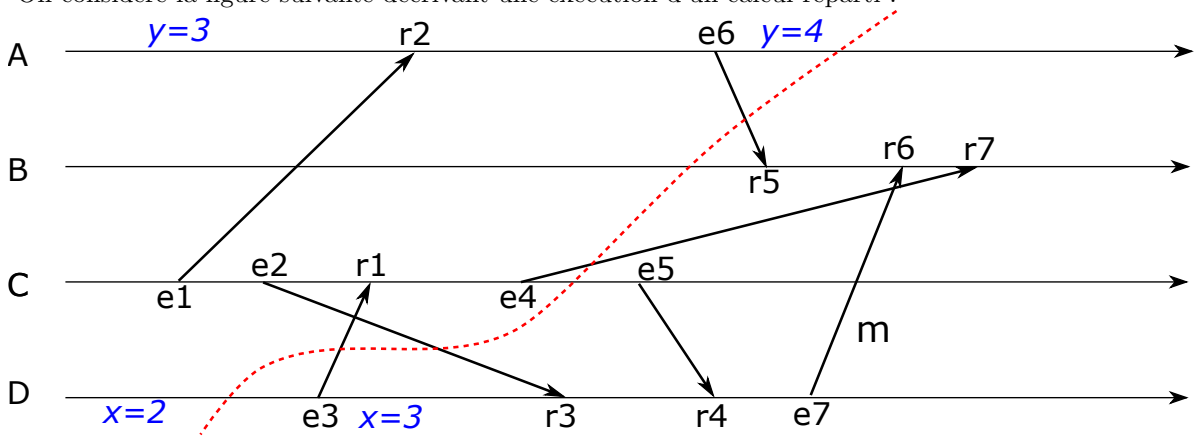
Systèmes et algorithmique répartis

ENSEEIHT/DIMA, master 2 Informatique et Télécommunication
1h45, documents autorisés

décembre 2015

1 Calcul réparti et causalité (7 points)

On considère la figure suivante décrivant une exécution d'un calcul réparti :



Questions (1 point par question)

1. Quel est le passé causal de r_4 ?
2. Quel est le futur causal de r_3 ?
3. Le chemin $e_1 \to r_2 \to e_5 \to r_4 \to e_7 \to r_6$ est-il un chemin causal liant e_1 à r_6 ?
4. La coupure indiquée est-elle cohérente ? Pourquoi ?
5. L'état global $x = 2 \wedge y = 4$ n'est pas survenu dans l'exécution présentée (x passe à 3 après l'événement e_3, et y passe à 4 après l'événement e_6). Cet état-il pourtant cohérent ?
6. Donner la valeur des horloges de Lamport pour tous les événements.
7. Déterminer l'histoire causale de m et en déduire si les délivrances r_4, r_5, r_6, r_7 respectent la causalité.

2 Problème (14 points)

Toutes les réponses doivent être justifiées. Un simple "oui", "non" ou "42" est considéré comme une absence de réponse.

Ce problème s'inspire du protocole Raft [1] pour construire un algorithme de réplication. Un ensemble de N serveurs est utilisé. Parmi ces serveurs, un « leader » unique est désigné et reçoit toutes les requêtes des clients. Ce leader ordonne les requêtes et les envoie vers les autres serveurs.

1. *In Search of an Understandable Consensus Algorithm*, Diego Ongaro and John Ousterhot, proceedings of Usenix Annual Technical Conference, 2014.

Le protocole proposé est basé sur deux phases indépendantes :

- l'*élection d'un leader*, lancée chaque fois que la panne du leader courant est présumée ;
- la *réplication*, réalisée par la réplication d'un *log* des écritures, contrôlée par le leader.

2.1 Hypothèses

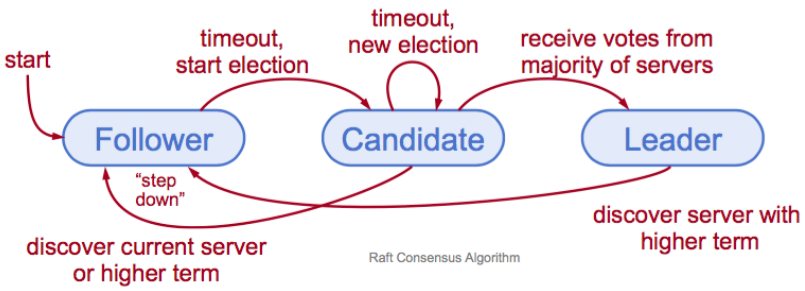
Les hypothèses de l'environnement considéré sont les suivantes :

- Le réseau est asynchrone, maillé.
- Les messages peuvent être perdus, mais non altérés.
- Les serveurs peuvent être sujets à des pannes d'arrêt : un serveur défaillant cesse de communiquer.
- Un serveur défaillant peut être relancé et rejoindre le groupe de serveurs. Dans ce cas, il reprend son exécution à partir de son état conservé en mémoire stable, précédant immédiatement la panne. Tout se passe donc comme si ce site n'avait pas progressé entre le moment de sa panne et le moment de sa reprise.
- Un serveur correct (c'est-à-dire non défaillant) s'exécute à une vitesse non nulle.
- La majorité des serveurs restent corrects.

2.2 L'élection

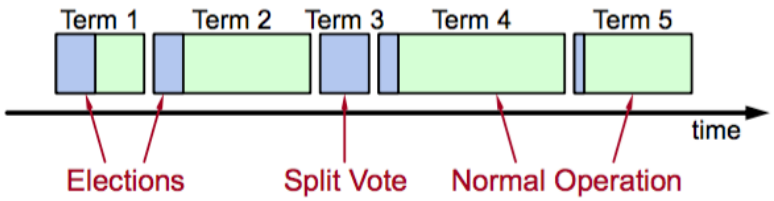
Un serveur (correct) peut se trouver dans l'un des trois états : leader, suiveur, ou candidat :

- le leader gère l'interaction avec les clients. Il ne doit jamais exister plus d'un leader ;
- un candidat est un serveur qui, ayant présumé une panne du leader courant a lancé une élection ;
- un suiveur répond aux requêtes du leader et des candidats.



Le temps est divisé en mandats successifs :

- les mandats sont identifiés par des entiers consécutifs croissants ;
- un mandat commence par une élection ;
- si l'élection aboutit, un leader unique est choisi, et ce leader coordonne la réplication. Si l'élection n'aboutit pas, un nouveau mandat est entamé. Le protocole doit garantir qu'il n'y pas plus d'un leader par mandat ;
- chaque serveur conserve le numéro de mandat (qu'il pense) courant, ce qui permet de de détecter les requêtes et les leaders obsolètes.



Lancement d'une élection Quand un suiveur suspecte une défaillance du leader, il lance une élection. Pour cela, si le suiveur n'a pas reçu de message du leader durant un laps de temps T_{max}, il présume une défaillance du leader. Afin d'éviter le lancement d'élection inutile, un leader correct envoie régulièrement des messages de contrôle (éventuellement vides), avec une période inférieure à T_{max}.

Questions

1. Vaut-il mieux que ce délai de garde soit différent pour chaque serveur, ou qu'il soit identique ?
2. Ceci forme un détecteur de faute basique. Quelle propriété de complétude a-t-il ?
3. Quelle propriété d'exactitude a-t-il ?

État de chaque serveur i

- `etat[i]` : $\in \{leader, candidat, suiveur\}$
- `mandat[i]` : le mandat courant de ce serveur est le plus grand mandat qu'il connaît ; si ce serveur est leader et qu'il n'y a pas d'élection en cours, son mandat courant est le plus grand des mandats de tous les autres serveurs.
- `vote[i]` : ce serveur a-t-il déjà donné son vote pour son mandat courant ? Remis à faux à chaque changement de mandat.

Un suiveur S_i lance une élection

- il incrémente `mandat[i]`, passe dans l'état candidat, arme le délai de garde d'élection T_{max}, vote pour lui-même, et envoie un message candidature(`mandat[i]`, S_i) à l'ensemble des serveurs ;
- Puis il attend :
  - d'avoir reçu une majorité de réponses positives des autres serveurs. Il devient alors le leader pour le mandat courant, et commence à émettre des messages de contrôle ;
  - ou de recevoir un message de contrôle d'un leader valide (c-à-d. d'un leader dont le mandat est supérieur ou égal au mandat courant). Il passe dans l'état suiveur et ajuste son mandat courant à celui de ce leader ;
  - ou de recevoir une réponse négative, avec un mandat supérieur ou égal à son mandat courant. Dans ce cas, il passe dans l'état suiveur et ajuste son mandat courant à celui de cette réponse ;
  - ou d'atteindre le délai de garde d'élection. Dans ce cas le candidat lance une nouvelle élection pour le mandat suivant.

Réception d'un message de candidature Lorsqu'un serveurs S_i reçoit un message `candidature(mandat,`S_j`)` :

- Si `mandat[i] > mandat`, alors il répond négativement avec son propre mandat ;
- S'il n'a pas encore voté lors de ce mandat, alors il répond positivement ;
- Sinon, il ignore la candidature.
- Dans tous les cas, si `mandat > mandat[i]` et que S_i était leader ou candidat, alors il redevient suiveur (il abandonne sa candidature s'il l'était pour ce mandat-là).

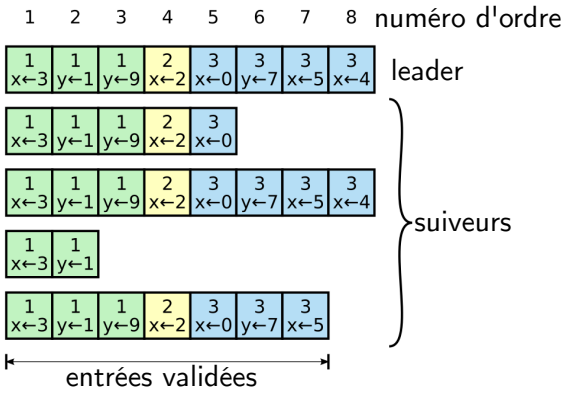
Questions

4. Dans quel état doit être un serveur qui démarre ou redémarre (après défaillance) ? Et avec quel mandat ?
5. Un suiveur peut-il recevoir des messages de candidature avec un numéro de mandat inférieur au sien ?
6. Un candidat peut-il recevoir des messages de candidature avec un mandat supérieur au sien ?
7. Montrer qu'un nouveau leader a un mandat nécessairement strictement plus élevé que celui du leader précédent.

8. Montrer que ce protocole ne peut pas aboutir à désigner plus d'un leader par mandat.
9. Dans le cas où seul le leader courant est défaillant, montrer qu'il est possible que le mandat suivant (= le mandat du leader défaillant plus 1) n'aboutisse pas, i.e. qu'il n'y a pas d'élu pour ce mandat.
10. En supposant qu'une majorité de serveurs restent corrects, est-il possible que le protocole d'élection n'aboutisse jamais ? Justifier la réponse par un argument (si non) ou un exemple (si oui).

2.3 La réplication

Toutes les requêtes d'un client sont envoyées au leader courant (le mécanisme permettant cela est sans importance). Le leader ordonne les écritures en leur attribuant un numéro d'ordre croissant, et les envoie aux autres serveurs. Chaque serveur acquitte la réception et quand une majorité de serveurs ont acquitté, le leader considère que cette écriture est validée et définitive. Chaque serveur (leader ou suiveur) range les écritures dans un *log*. La correction de l'algorithme est basée sur le fait que les logs des suiveurs sont toujours un préfixe de celui du leader, cf figure. Chaque case contient le numéro de mandat (on en est donc au troisième leader) et l'écriture effectuée.



Chaque message contient :

- le mandat courant du leader, ce qui permettra de détecter et d'ordonner correctement des vieilles requêtes ;
- le plus grand numéro que le leader a validé, ce qui permet aux suiveurs de valider à leur tour ces écritures. Cela est nécessaire pour changer de leader.

Question

11. Quand un client demande à lire x, si le leader lui donne la plus récente écriture de son log (4 sur la figure), expliquer quelle est la cohérence obtenue : en absence de panne / en cas de panne.
12. Quand un client demande à lire x, si le leader lui donne la plus récente écriture validée (5 sur la figure), expliquer quelle est la cohérence obtenue : en absence de panne / en cas de panne.
13. Montrer, éventuellement en s'aidant de la figure, qu'en cas de panne du leader, seuls certains suiveurs sont éligibles si l'on veut maintenir la cohérence du log.
14. Modifier en conséquence l'algorithme du choix du leader.