SCIENCES SORBONNE UNIVERSITÉ

TD 9 - Terminaison

Avril 2020

Contexte

- ✯ il y a N sites qui ont tous des noms distincts, et savent que leurs noms sont distincts. De plus, N est connu de tous les sites.
- ✯ les communications sont fiables : tout message émis est délivré au bout d'un temps arbitraire mais fini ;
- ✯ les liaisons sont bidirectionnelles, le graphe de communication est fortement connexe ;

Exercice(s)

Exercice 1 – L'algorithme de Huang (1988)

Cet algorithme généralise l'algorithme de Rana (1980) et permet de détecter la terminaison sur une topologie de réseau quelconque. Les sites disposent de deux primitives de communication :

- `send(<type, contenu, horloge, emetteur, recepteur>)` : envoie un message de l'émetteur au récepteur (NB : le champ `contenu` n'est pas utilisé lors de l'envoi d'un message de contrôle, il n'est utile que pour les messages de l'application ;
- `broadcast(<type, horloge, emetteur>)` : permet à l'émetteur de diffuser une information (de contrôle) à tous les sites.

L'algorithme repose sur l'utilisation d'horloges logiques scalaires. Chaque site gère une horloge lui permettant de mémoriser la date de fin supposée de l'application. Cette date est mise à jour :

- chaque fois que le site détecte sa terminaison locale : il est inactif et aucun des messages qu'il a envoyés n'est en transit ; cette condition peut être réalisée plusieurs fois si le site est réactivé en recevant un message de l'application ;
- lorsqu'un autre site lui annonce sa terminaison, si l'horloge portée par le message est postérieure à la date de fin qu'il connaît. Pour avoir un ordre total sur les horloges, on utilise le couple *(valeur d'horloge, site)*. Chaque site mémorise donc l'identité du site qui, à sa connaissance, s'est terminé le dernier.

Le principe est le suivant :

- un site qui détecte sa terminaison locale incrémente son horloge et diffuse l'information sur sa terminaison à tous les autres sites ;
- un site qui reçoit une annonce de terminaison met à jour son horloge et ne répond (envoi d'un message `agr`) que s'il est "localement terminé" et si l'horloge contenue dans le message est supérieure à son horloge locale ;
- les messages de l'application dont on détecte la terminaison ne portent pas d'horloge, seuls les messages de l'algorithme de terminaison en ont une.

Question 1

A quel moment un site peut-il décider que l'application est terminée ?

Solution:
Un site qui reçoit un message `term` ne répond que s'il est localement terminé et si son horloge est inférieure à l'horloge portée par le message. Si tous les sites répondent, c'est donc qu'ils étaient tous localement terminés à une date antérieure à l'horloge portée par le message. Un site qui reçoit de tous les autres une réponse à son message $\langle term, h\rangle$ peut donc supposer que l'application s'est terminée à la date `h`.

Cependant, il faudra s'assurer que si un site a été réactivé après avoir répondu, alors au moins un site n'a pas répondu.

Fin solution

Question 2

Proposez un mécanisme permettant à chaque site de savoir si un message envoyé a été reçu. Donnez les variables nécessaires à la gestion de ce mécanisme.

Solution:
Il faut gérer un mécanisme d'acquittement. Le site i gère une variable locale `unack` qui comptabilise le nombre de messages envoyés par i et non acquittés. Lorsque `unack = 0`, tous les messages de i ont été acquittés donc *a fortiori* reçus. On note `h[i]` l'horloge scalaire du site i.

Fin solution

Question 3

Écrivez les primitives exécutées par le site i

- lorsqu'il envoie un message (de l'application),
- lorsqu'il reçoit un message (de l'application).

Solution:
Les procédures exécutées sont :

```
envoyer(<mes, cont, -, i, j>) {
   /* Executable uniquement si i est actif */
   unack++;
   send(<mes, cont, -, i, j>);
}

recevoir(<mes, cont, -, j, i>) {
   si (etat == inactif) {
      etat = actif ;
   }
   send(<ack, -, h[i], i, j>);
}
```

Fin solution

Question 4

A quel moment le site i peut-il lancer une détection de la terminaison ?

Écrivez les primitives exécutées par le site i lorsqu'il passe de l'état actif à l'état inactif, et lorsqu'il reçoit un acquittement.

Solution:
Lorsqu'il devient "localement terminé", ce qui peut se produire soit au moment de la désactivation, soit lorsqu'il reçoit l'acquittement correspondant à son dernier message en transit. Il doit alors tester si c'est aussi le cas des autres sites pour savoir si l'application est terminée. On note `term` le type des messages de détection de la terminaison. La variable locale `last` mémorise l'identité du site dont la terminaison est la plus récente (à la connaissance de i). On ajoute aussi une variable `hfin` (en réalité redondante avec `h[i]`) pour une meilleure lisibilité. Le couple `(last, hfin)` représente la date à laquelle i suppose que l'application s'est terminée (mise à jour à chaque nouvelle information sur la terminaison).

```
desactivation(i) {     /* Executable uniquement si i est actif */
   etat(i) = inactif;
   si (unack == 0) {
        h[i]++;
        last = i;
        hfin = h[i];
        nbagr = 0;
        broadcast (<term, h[i], i>);
    }
}

recevoir(<ack, -, h, j, i>) {
    unack--;
    h[i] = max(h, h[i]) ;
    si (unack == 0 && etat(i) == inactif) {
        h[i]++;
        last = i;
        hfin = h[i];
        nbagr = 0;
        broadcast (<term, h[i], i>);
    }
}
```

Fin solution

Question 5

Écrivez la primitive exécutée par le site i lorsqu'il reçoit un message de type `term`.

Solution:

Lorsque le site reçoit un message `term` alors qu'il n'est pas encore terminé, il met à jour son horloge par application du max pour être sûr que sa terminaison aura une date postérieure à celle contenue dans le message. S'il est terminé, il compare les horloges pour décider s'il répond ou non. Si c'est le cas, il met son horloge à jour pour qu'elle stocke la date de terminaison la plus récente qu'il connaît, qui est la date contenue dans le message.

```
recevoir(<term, h, j>) {
    si ( (etat(i) == actif) || (unack > 0)) {
        h[i] = max(h, h[i]);
    }
    sinon {
        si ( (h, j) > (hfin, last) ) {
            h[i] = h;
            hfin = h;
            last = j ;
            envoyer (<agr, -, h, i, j>);
        }
    }
}
```

Fin solution

Question 6

Écrivez la primitive exécutée par le site i lorsqu'il reçoit un message de type `agr`.

Solution:

Un site a pu lancer plusieurs messages de terminaison : il était localement terminé (envoi d'un premier message), a été réactivé puis s'est à nouveau localement terminé (envoi d'un deuxième message). On ne comptabilise donc les

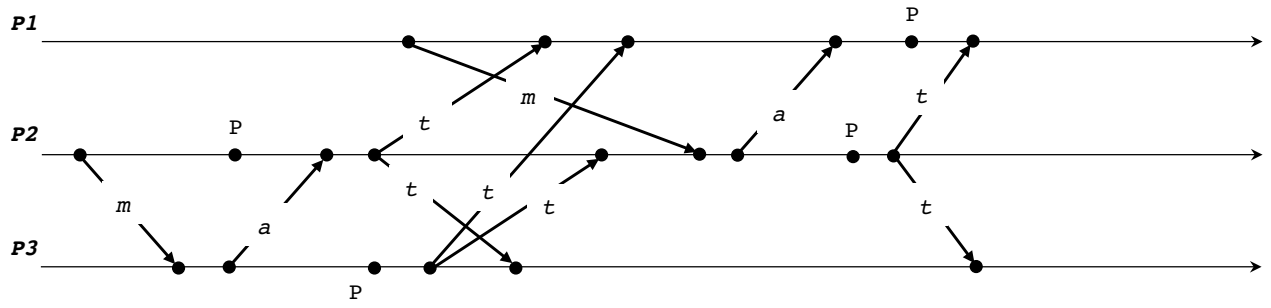

FIGURE 1 – Exécution sur 3 sites

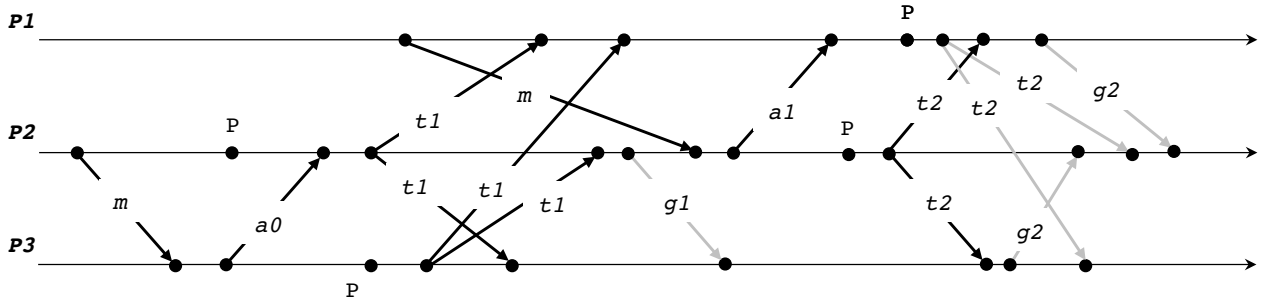

FIGURE 2 – Exécution complète sur 3 sites

accords que s'ils portent l'horloge `hfin`. Si `hfin` a été modifiée depuis la diffusion de l'annonce de terminaison, alors il y a eu une annonce plus récente (de la part de i ou d'un autre site) et le message reçu correspond à une annonce qui n'aboutira pas, il est donc inutile de le comptabiliser.

```
recevoir(<agr, -, h, j, i>) {
   si (h == hfin) {
      nbagr++;
      si (nbagr == N-1) {
         annoncer la terminaison ;
      }
   }
}
```

Fin solution

Question 7

Complétez le schéma d'exécution de la Figure 1 en ajoutant les messages manquants et les horloges des messages (les états P représentent les états ou un site devient inactif, les lettres indiquent les types de messages : `m` pour les messages de l'application, `a` pour `ack` et `t` pour `term`).

Solution:

La Figure 2 représente l'exécution complète. Les messages `agr` ont été représentés par `g`. Les messages de l'application ne portent pas d'horloge. L'horloge d'un site est incrémentée par application de la règle du max et lorsque le site se termine localement. Cela permet de garantir que tout site a une date de terminaison locale strictement supérieure à toute date de terminaison qu'il connaît au moment où il se termine.

Fin solution

Question 8

Quelles sont les modifications à apporter pour optimiser l'algorithme si la structure de contrôle est un anneau ?

Solution:
On n'effectue pas de diffusion sur un anneau. Le site i envoie un message ⟨term, h, i⟩ à son voisin. Celui-ci fait circuler le message s'il est localement terminé et si h est supérieure à la date supposée de terminaison de l'application qu'il connaît. Un site qui voit revenir son message sait donc qu'il est le dernier à s'être localement terminé et il peut donc annoncer la terminaison.
Fin solution