Algorithmique Distribuée : Examen

Stéphane Devismes

Lisez attentivement ce qui suit :

— Aucun document n'est autorisé.

— Le barême est donné à titre indicatif uniquement. Il est susceptible de changer !

1 Détecteur de défaillances (5 points)

Nous rappelons que tout détecteur de défaillances *finalement parfait* $\diamond\mathcal{P}$ satisfait la **complétude forte** et l'**exactitude finalement forte** :

Complétude forte : Tout processus qui tombe en panne finit par être suspecté en permanence par *tous* les processus corrects.

Exactitude finalement forte : Il existe un temps à partir duquel plus aucun processus correct n'est suspecté par un processus correct.

On suppose un réseau **complet** sujet à des pannes définitives **initiales** de processus, c'est-à-dire que tout processus qui tombe en panne n'a jamais participé à aucun calcul. De plus, les liens de communications sont sujets à **des pertes équitables**.

Le but de cet exercice est d'écrire l'algorithme d'un détecteur de défaillances finalement parfait.

Questions.

1. Sous les hypothèses données, quelle propriété intéressante sur un processus p, un processus peut-il déduire s'il reçoit un message provenant de p ? **(0,5 point)**
2. Sous les hypothèses données, comment peut-on s'assurer qu'un processus correct finisse par ne plus être suspecté par les autres ? **(0,5 point)**
3. Écrivez un algorithme réalisant le détecteur de défaillance **finalement parfait** dans le système présenté ci-dessus. **(2 points)**
4. Prouvez la correction de votre algorithme. **(2 points)**

2 Autostabilisation (15 points)

2.1 Comprendre un algorithme autostabilisant (6 points)

Ci-dessous nous rappelons le premier algorithme de circulation de jeton proposé par Dijkstra.

Nous considérons un anneau orienté de $n > 0$ processus : p_0, ..., p_{n-1}. Chaque processus p_i peut lire l'état de son prédécesseur p_{i-1} (les calculs sur les indices sont modulo n bien entendu !). Tous les processus ont le même programme, sauf p_0. Ce dernier est appelé la *racine*. Par suite, les autres sont appelés processus *nonracines*. On souhaite faire circuler un unique jeton dans le réseau. Ainsi la spécification de notre problème est la suivante : il existe un unique jeton dans le réseau et chaque processus détient le jeton infiniment souvent.

Tous les processus ont une unique variable v dont le domaine est $\{0, \ldots, K-1\}$ avec $K \geq n$. Un processus p_i détient un jeton si et seulement si il satisfait le prédicat $Token(p_i)$.

Le programme de chaque processus consiste en une unique règle :

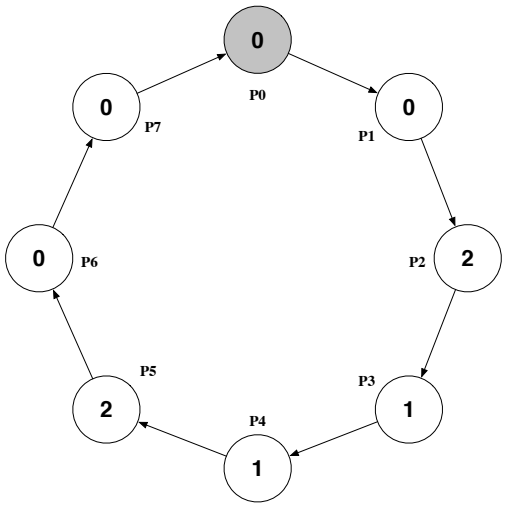

FIGURE 1 – Une configuration initiale du premier algorithme de Dijkstra.

Programme de la racine p_0 :

$$Token(p_0) \mapsto v_{p_0} \leftarrow (v_{p_0} + 1) \bmod K, \text{ où } Token(p_0) \equiv (v_{p_0} = v_{p_{n-1}})$$

Programme de chaque processus p_i nonracine :

$$Token(p_i) \mapsto v_{p_i} \leftarrow v_{p_{i-1}}, \text{ où } Token(p_i) \equiv (v_{p_i} \neq v_{p_{i-1}})$$

Questions.

5. Donnez la trace d'exécution des 8 premiers pas de calculs du premier algorithme de Dijkstra à partir de la configuration initiale de la figure 1 en supposant un démon synchrone. **(4 points)**
6. Démontrez qu'il n'existe pas de configuration sans jeton. **(2 points)**

2.2 Écrire un algorithme autostabilisant (9 points)

On considère un réseau en arbre bidirectionnel, enraciné en R et orienté. C'est-à-dire, chaque processus p peut lire l'état de n'importe quel voisin. De plus, chaque processus nonracine p connaît le numéro de canal de son père dans l'arbre : ce numéro de canal est noté $père_p$. Nous désignons aussi par $\mathcal{N}_p$ l'ensemble des numéros de canaux des voisins de p.

Un exemple d'arbre est donné dans la figure 2.

On suppose que chaque processus p connaît son numéro de canal dans l'ensemble $\mathcal{N}_q$ de chacun de ses voisins q. Ainsi, p pourra tester si, par exemple, le pointeur $père_q$ de son voisin q le désigne comme son père dans l'arbre. Dans la suite, un tel test sera simplement écrit $père_q = p$.

Question.

7. Dans un arbre, l'ensemble des fils d'un nœud correspond à l'ensemble de ses voisins autres que son père. Définissez formellement $Children_p$, l'ensemble des fils de p. **(1 point)**
8. Comment un processus peut évaluer qu'il est une feuille de l'arbre ? **(0,25 point)**

On souhaite écrire un algorithme autostabilisant silencieux qui calcule le nombre de feuilles dans l'arbre.

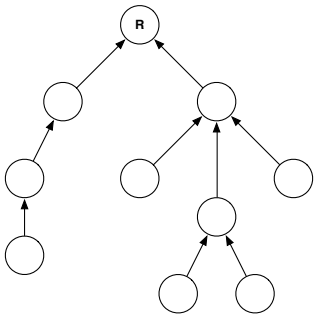

FIGURE 2 – Exemple d'arbre : la racine est étiquetée R ; les flèches représentent les liens « père ».

Questions.

9. Rappelez la définition d'une configuration *terminale*. **(0,25 point)**
10. Rappelez la définition d'un algorithme *silencieux*. **(0,5 point)**
11. Quelles sont les deux étapes principales nécessaires pour prouver qu'un algorithme est auto-stabilisant et silencieux ? **(1 point)**

Pour écrire l'algorithme, vous avez besoin de deux variables entières par processus p :

— Dans Sub_p vous devez évaluer le nombre de feuilles existant dans le sous-arbre de p.
— Dans F_p vous devez stocker le nombre total de feuilles dans l'arbre.

Questions.

12. Dessinez la configuration terminale que vous devez obtenir en exécutant l'algorithme sur le réseau donné en figure 2 ? **(1 point)**
13. Donnez les règles de l'algorithme. **(3 points)**
14. Quel est le temps de stabilisation en rondes de votre algorithme ? Proposez un pire cas. **(2 points)**