Aix*Marseille université
Socialement engagée

Année 2023-24

Domaine Sciences et Technologies
Master informatique

Fiabilité Logicielle : TP 4
Code UE : SINBU22

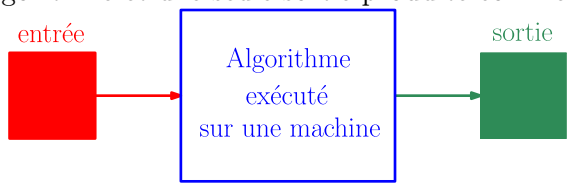
TP 4 : RAFT (partie 1/3)

1 Introduction

1.1 Problématique

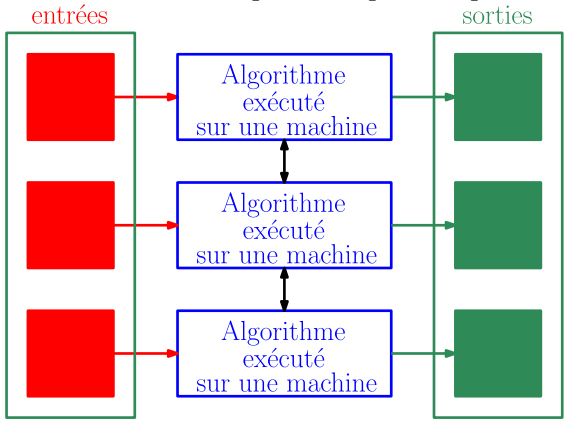
De nos jours, il est très courant de se connecter à un service sur internet et d'utiliser comme si celui-ci était hébergé sur un seul serveur, c'est-à-dire une seule machine physique. Par exemple, lorsqu'on utilise un réseau social, les modifications que l'on fait sur notre compte sont visibles de manière quasi instantanée quelle que soit la manière dont on se connecte. Il est clair qu'il est impossible d'héberger tout le trafic d'un service mondial populaire (plusieurs centaines de millions d'utilisateurs quotidiens ou plus) sur une seule machine physique. Ces services sont donc hébergés sur plusieurs machines parfois très éloignées qui doivent régulièrement communiquer entre elles pour maintenir la cohérence du contenu hébergé. Cette problématique est connue sous le nom du problème du consensus. Une solution à ce problème consiste en un algorithme qui sera exécuté sur chaque machine ou nœud (*node*) d'un réseau.

Un tel algorithme dit *distribué* décrit le fonctionnement d'un système informatique composé de plusieurs unités de calcul reliées par un réseau de communication. Un algorithme distribué est fondamentalement différent des algorithmes que vous avez croisés jusqu'à lors dans vos études. Pour un algorithme classique, aussi appelé *séquentiel*, il y une seule entrée (les données du problème à résoudre), une seule machine sur laquelle est exécuté l'algorithme et une seule sortie produite comme l'indique le diagramme ci-dessous.



La spécification d'un algorithme séquentiel détermine quelles sont les sorties valides en fonctions de l'entrée donnée. Par exemple, pour un algorithme de tri de tableau de données, toute sortie contenant les mêmes éléments que les données d'entrées qui sont rangés dans un ordre croissant selon le comparateur fourni en entrée est valide.

Pour un algorithme distribué, il y a généralement une entrée par machine (nœud) du système, plusieurs machines et une sortie produite par chaque machine du système comme l'indique le diagramme ci-dessous.



Chaque machine exécute le même algorithme, mais pas forcément avec les mêmes informations. De manière similaire aux algorithmes séquentiels, la spécification d'un algorithme distribué détermine quelles sont les en-

sembles de sorties valides en fonctions de l'ensemble des entrées données au système (pas forcément la même entrée pour toutes les machines).

Les machines peuvent communiquer en envoyant et en recevant des messages. La difficulté des problèmes distribués provient du fait que dans un système comportant de nombreuses machines s'envoyant de nombreux messages, des pannes vont probablement survenir. Ces pannes peuvent prendre la forme de ralentissement du réseau de communication, de pertes de messages, d'une coupure du réseau de communication ou bien encore un certain nombre de machines qui ne fonctionnent plus pendant une période donnée. C'est généralement l'incertitude au sujet des pannes qui rend difficile, voire impossible la résolution de certaines tâches distribuées.

Dans ce TP, on s'intéressera à deux problèmes qui occupent une place importante en algorithmique distribuée : le problème du consensus et celui de l'élection.

1.2 Problème du consensus

Le problème du consensus est un problème fondamental pour lequel chaque nœud doit choisir une valeur parmi celles initialement présentes dans les valeurs initiales. Intuitivement, résoudre le problème du consensus revient donc à se mettre d'accord sur une valeur et donc s'accorder sur des données.

Afin de définir formellement le cadre du problème, il est important de décrire avec précision ce que veut dire choisir une valeur. On dit qu'un nœud *décide* lorsqu'une valeur finale pour le calcul est donnée par celui-ci. Une fois qu'il a décidé une valeur, un nœud ne peut donc plus changer d'avis. Une spécification possible du consensus est la spécification suivante :

Spécification du consensus

Étant donnés des processus avec chacun une valeur initiale dans un ensemble U :

— **Accord** : Tous les processus qui décident une valeur décident la même valeur.
— **Intégrité** : La valeur décidée est une valeur initialement proposée.
— **Terminaison** : Tous les processus **non défaillants** décident une valeur.

L'algorithme RAFT que l'on va étudier résout ce problème de manière itérée.

1.3 Problème de l'élection

Un autre problème, important dans le cadre du distribué, est celui de l'élection. Intuitivement, l'élection consiste à choisir un des nœuds du système qui aura un état spécial appelé ÉLU. C'est une tâche très importante et qui est souvent utilisée dans la résolution d'autres tâches distribuées. Cela sera d'ailleurs le cas dans l'algorithme de consensus qu'on va étudier.

Une spécification possible de l'élection est la spécification suivante :

Spécification de l'élection

Un nœud et un seul, doit terminer dans l'état `ÉLU`.

On peut aussi rajouter la propriété suivante :

Les autres nœuds doivent terminer dans l'état `NON_ÉLU`.

2 Mise en place

2.1 Article

Pour ce TP, nous allons étudier un algorithme de consensus appelé RAFT. La première étape du TP consiste donc à télécharger l'article décrivant l'algorithme au lien suivant : `https://raft.github.io/raft.pdf`.

Diego Ongaro and John K. Ousterhout, *In Search of an Understandable Consensus Algorithm.* USENIX Annual Technical Conference 2014 : 305-319

Lire le résumé (*abstract*) et l'introduction de cet article. En préliminaire, commencer par suivre ce tutoriel afin de bien comprendre la problématique de l'algorithme RAFT.

2.2 Simulateur

Les auteurs de l'article proposent un simulateur `raftscope`, en javascript, de cet algorithme pour 5 nœuds.

> **Question 1 :** Expliquez si 5 processus permettent de vérifier la correction de l'algorithme.

Installez le simulateur localement avec la commande suivante :

```
git clone https://github.com/ongardie/raftscope.git
```

Placez-vous dans le répertoire de votre dépôt cloné (`raftscope`), puis installez les dépendances avec la commande suivante :

```
git submodule update --init --recursive
```

Ouvrez le fichier `index.html`. Vous devriez voir s'afficher l'interface du simulateur avec les cinq nœuds.

2.3 Contrôle

> **Question 2 :** Décrivez ce qui permet de contrôler et voir les étapes de l'algorithme dans le simulateur.

3 Partie 1 : Élection

3.1 Démarrage et Élection

Lancer la simulation et observer :

> **Question 3 :** Décrivez ce qu'il se passe. Comment sont disposés les processus ? Qui peut communiquer avec qui ?

> **Question 4 :** Un processus est distingué (cercle sur le schéma, en rouge sur le tableau), pour quelle raison ?

> **Question 5 :** Est-ce toujours le même processus qui est distingué ? pendant l'exécution ? au redémarrage de la simulation ?

3.2 Manipulation des Nœuds

En cliquant sur un nœud, on peut voir son état en détail et également modifier son comportement.

Question 6 : À quoi correspondent les boutons :
— **stop**
— **resume**
— **restart**
— **time out**
— **request**

Question 7 : Comment peut-on simuler la perte d'un message ?

3.3 Robustesse

Question 8 : Par des manipulations successives sur tous les processus (le processus Élu, les autres), vérifier qu'il n'y a toujours qu'un seul processus distingué (c'est-à-dire Élu). Notez les scénarios (c'est-à-dire la suite de manipulations sur les processus) et les résultats correspondants.

4 Compte-Rendu

Il est attendu un compte rendu détaillé pour ce TP qui est à faire seul ou en binôme (mêmes équipes que les précédents TP). Ce compte rendu contiendra vos réponses aux questions ainsi qu'éventuellement des expérimentations supplémentaires que vous auriez faites. Le compte rendu devra être un unique fichier `pdf` à déposer sur AMETICE avant le mardi 30 janvier 2024 à 23h59. Le fichier devra préciser le nom complet ou les deux noms complets des personnes ayant participé à sa rédaction. Un point important sera accordé aux aspects explicatifs et synthétiques dans votre compte rendu.