TP 5
Algorithme d'élection d'un leader

3 heures

Prérequis : TP4 terminé

1 Algorithme de Chang et Roberts

L'objectif de ce TP est d'implémenter l'algorithme d'élection de leader de *Chang et Roberts*[1] au dessus de la version simplifiée du réseau Chord implémentée durant la séance de TP précédente. Le principe de l'algorithme d'élection de leader consiste à échanger des messages entre les différents pairs du réseau afin qu'ils se mettent d'accord sur l'identifiant d'un pair qui sera désigné leader. Le leader désigné sera connu de tous les pairs ayant participé à l'élection et pourra être utilisé pour effectuer une tâche.

Commencez par copier les sources du TP précédent dans un nouveau projet et travaillez sur cette copie.

1.1 Élection d'un leader

La première étape de l'algorithme de *Chang et Roberts* consiste à trouver un leader. L'algorithme est tel que le leader sera toujours le pair ayant l'identifiant le plus grand.

1. Ajoutez une méthode *startElection* dans l'interface *Peer* ;
2. Ajoutez deux variables d'instance dans la classe *PeerImpl* nommées *leaderId* et *participant* qui sont respectivement de type *Identifier* et *boolean* ;
3. Ajoutez dans l'interface *Peer* les getters et setters associés aux champs introduits précédemment ;
4. Créez une interface *Message* qui déclare la signature d'une méthode nommée *handle* prenant en paramètre un objet de type *Peer* ;
5. Ajoutez une méthode *receive* à l'interface *Peer*, prenant un paramètre de type *Message*. L'implémentation de cette méthode doit appeler la méthode *handle* du message passé en paramètre afin de traiter le message reçu ;
6. À partir de l'interface et de la méthode introduite précédemment il est possible de définir, d'envoyer et de faire réceptionner par des pairs différents types de messages. Il faut maintenant créer une classe nommée *ElectionMessage* qui va définir le comportement à appliquer pour élire un leader. Cette classe contient une variable d'instance qui correspondant à l'identifiant du leader désigné. Cette variable d'instance sera initialisée à partir d'une valeur donnée lors de la construction d'un objet de type *ElectionMessage* ;

1. `http://en.wikipedia.org/wiki/Chang_and_Roberts_algorithm`

7. Définissez les implémentations des méthodes ajoutées dans l'interface *Peer*. La méthode *startElection* doit déclarer le pair courant comme participant à l'algorithme d'élection puis envoyer un message de type *ElectionMessage* (créé en passant l'identifiant du pair courant au constructeur) à son successeur ;

8. Quand un pair reçoit un message de type *ElectionMessage*, il doit comparer l'identifiant du leader contenu dans le message reçu avec le sien.

   (a) Si l'identifiant contenu dans le message reçu est plus grand que celui du pair p ayant reçu le message, alors le message est envoyé au successeur de p ;

   (b) Si l'identifiant contenu dans le message reçu est plus petit que celui du pair p ayant reçu le message et que p n'a pas encore participé au processus d'élection, p se déclare comme ayant participé au processus d'élection, modifie l'identifiant contenu dans le message pour y mettre son identifiant et envoie le message modifié à son successeur ;

   (c) Si l'identifiant contenu dans le message reçu est plus petit que celui du pair p ayant reçu le message et que p a déjà participé au processus d'élection le pair ignore le message ;

   (d) Si l'identifiant contenu dans le message reçu est égal à l'identifiant du pair ayant reçu le message, alors le pair est déclaré leader et met à jour sa variable d'instance *leaderId* avec son identifiant ;

9. Modifiez la classe *Main* copiée du TP4 afin de faire réceptionner, par le successeur d'un pair choisir aléatoirement, un message de type *ElectionMessage* ;

10. Ajoutez des traces afin de vérifier les différentes étapes de l'algorithme. Les informations affichées devraient donner quelque chose de similaire à :

```
Election started from 200
Handling election message on peer with id 300, 200.compareTo(300)=-100, participate? false
Handling election message on peer with id 400, 300.compareTo(400)=-100, participate? false
Handling election message on peer with id 0, 400.compareTo(0)=400, participate? false
Handling election message on peer with id 100, 400.compareTo(100)=300, participate? false
Handling election message on peer with id 200, 400.compareTo(200)=200, participate? true
Handling election message on peer with id 300, 400.compareTo(300)=100, participate? true
Handling election message on peer with id 400, 400.compareTo(400)=0, participate? true
Leader found: 400
```

1.2 Diffusion de l'identifiant de l'élu

Une fois le leader trouvé, il faut prévenir les autres pairs qui ont pas forcemment connaissance du nouveau leader. Pour cela il vous faut :

1. Créer un nouveau message nommé *ElectedMessage* prenant l'identifiant du leader en paramètre ;

2. Envoyer ce message au successeur du pair déclaré comme leader ;

3. Lors de la réception d'un message de type *ElectedMessage*, si l'identifiant contenu par le message est différent de celui du pair ayant reçu le message, alors le pair courant met à jour son champs *leaderId* avec l'identifiant contenu dans le message puis envoie le message à son successeur. Dans le cas contraire le message est ignoré ;

4. Ajoutez les traces nécessaires pour suivre l'exécution de l'algorithme. Vous devriez obtenir quelque chose de similaire à :

```
Election started from 200
Handling election message on peer with id 300, 200.compareTo(300)=-100, participate? false
Handling election message on peer with id 400, 300.compareTo(400)=-100, participate? false
Handling election message on peer with id 0, 400.compareTo(0)=400, participate? false
Handling election message on peer with id 100, 400.compareTo(100)=300, participate? false
Handling election message on peer with id 200, 400.compareTo(200)=200, participate? true
Handling election message on peer with id 300, 400.compareTo(300)=100, participate? true
Handling election message on peer with id 400, 400.compareTo(400)=0, participate? true
Leader found: 400
Peer with id 0 notified about new leader: 400
Peer with id 100 notified about new leader: 400
Peer with id 200 notified about new leader: 400
Peer with id 300 notified about new leader: 400
```