Théorie d'application répartie

Louis Auneau, Sélim Dahmani, Jeremy Mathey

March 2016

1 Implémentations de base

Nous avons choisi de créer 2 classes qui géreront les principales interactions dans le réseau :

— `NetworkManager` : Cette classe dont toutes ses méthodes sont statiques sert à notamment se connecter à un réseau et à envoyer des messages. Cela sera principalement la "bouche" de la machine.

— `NetworkListener` : Cette classe va écouter les messages traversant sur le réseau. Selon les messages reçus il va déclencher des méthodes (par exemple : pour ajouter un pair comme successeur, répondre à une machine qui veut s'insérer dans le réseau etc...) pour gérer les données arrivant par messages.

La classe `Pair` s'occupe des attributs que possède un pair, c'est-à-dire : le successeur, le prédecesseur, les *fingers*. Cette classe est tout d'abord créée dans le `main` avant de s'insérer dans le réseau. Grâce à cette classe on pourra définir à le destinataire (soit le successeur, soit le prédecesseur ou soit l'un des fingers) à qui on doit envoyer le message ou faire passer un message, efficacement. Cette classe est en effet chargé d'optimiser l'envoi des messages via les fingers/successeurs/prédecesseurs afin d'envoyer un message au pair le plus proche de la cible d'après un calcul très simple utilisant les modulos sur les hashs.

Tous les messages que nous avons défini possède une "en-tête" correspondant à une tâche (ou demande) bien précise. Elles sont défini dans une enumération dans la classe Message. Cela facilitera le changement de message.

2 Implémentations de l'extention

Nous avons choisi comme extension la gestion d'un départ (attendu ou non). Nos choix d'implémentation sont simples :

2.1 Départ attendu

Si un pair veut se déconnecter "proprement", il envoie un message `a+:valeurDuHashDuPair` au `WelcomeServer` qui supprimera le pair dans sa table de routage. Et le notifie à ses pairs qu'il va se déconnecter afin qu'ils les autres pairs refassent leurs tables de routage.

2.2 Départ inattendu

Si un pair se déconnecte (que l'on va nommer pairX pour l'explication) sans le notifier au WelcomeServer, on compte sur les pairs pour le notifier. En effet, la notification de la part des pairs se fera lorsqu'une tentative de communication à un pair mal déconnecté générera une exception (IOException). Dans le catch de cette exception nous déclencherons une méthode qui notifierera le WelcomeServer et les autres pairs présents sur le réseau, que le pairX est manquant. Lorsque les pairs vont recevoir cette notification, ils vont vérifier si pairX est présent dans leur table de routage. Si c'est le cas, les pairs conernés vont enclencher une procédure (via envoi de message) pour refaire remplacer pairX dans leur table de routage par un autre pair présent dans le réseau.

2.3 Autre choix possible

Nous aurions très bien pu choisir une autre stratégie pour trouver des pairs qui se sont mal déceonnecté. En effet, nous aurions pu faire en sorte en définissant un intervalle de temps N que tout les N secondes un message circule dans le réseau pour vérifier la présence de tout les pairs. Cela fonctionnerais comme une fiche d'appel de

pair en pair. Si un pair ne répond pas à ce message alors il est dit “absent” et notifié au WelcomeServer et aux autres pairs.