Théorie des systèmes répartis

Projet pair à pair

Nicolas BLIN - Bryan AUDIC - Corentin GOINARD - Simon ROUSSEAU
INFO 2 - Groupe 1

Le but de ce projet est de réaliser un protocole de réseaux pair à pair en se basant sur le modèle Chord étudié en cours.

Sommaire :

I - Description du projet

Dans le cadre du module "Théorie des systèmes répartis", nous avons été amenés à développer un projet pair à pair qui implémenterait le protocole Chord.

Pour réaliser ce projet, nous avons choisi d'utiliser JAVA, un langage maîtrisé par les membres du groupe qui nous offre une utilisation simplifiée des sockets et des threads en général.

Nous avons eu à notre disposition différents outils pour réaliser notre projet, fournis par notre enseignant. Nous disposions en outre de :

-Serveur d'accueil : classe implémentant le WelcomeServer de JAVA, nous permettait d'intégrer un pair à notre réseau.
-Serveur de hachage : nous octroyait des hashs pour notre serveur
-Monitor : permet l'interaction de notre serveur à partir d'un terminal, pour permettre d'effectuer des vérifications.

Nicolas BLIN - Bryan AUDIC - Corentin GOINARD - Simon ROUSSEAU

INFO 2 - Groupe 1

II - Explication de l'implémentation

a) Implémentation réalisée

Pour l'implémentation de ce projet, nous avons développé une classe pair qui avait pour objectif de se connecter au WelcomeServer afin de pouvoir connaître l'adresse du pair que nous devions contacter pour entrer dans le réseau. Pour mettre ceci en place, nous avons 4 attributs dans la classe pair: Notre IP, notre Hash, l'IP du successeur et le hash du successeur. Pour la partie communication,nous avons utilisé des sockets. Une première pour se connecter au WelcomeServer et récupérer sa réponse.
Dans le cas où nous avons "yaf", nous initialisons l'IP et le Hash du successeur par nos propres paramètres, ainsi nous créons un thread afin d'écouter d'éventuelles connexions d'autres pairs.
Dans le cas où nous avons une adresse IP, il faut alors se connecter à celle-ci au moyen d'une socket et transmettre une requête avec notre Hash et l'IP. Puis il faut créer un ServerSocket afin d'être à l'écoute de la réponse, puis on l'analyse et on en extrait les données tel que l'IP et le Hash de notre successeur.

Puis après la réception du message du serveur et du positionnement du pair entrant sur le réseau, celui-ci crée un thread lui permettant d'être récepteur de messages. Pour cela, il utilise une instance de PeerRunnable implémentant Runnable, ainsi nous pouvons utiliser la méthode run. Celle-ci va alors créer un ServerSocket, lui permettant de pouvoir accepter toute socket.

Lorsqu'il y a l'ouverture d'une connexion, nous analysons le message envoyé selon le contenu:

REQUETE_DEMANDE, celle-ci a pour but d'envoyer son propre hash.
REQUETE_MODIF_SUCCE, ce message a pour objectif de faire changer le successeur du pair qui le reçoit.
REQUETE_SUCCE, ici c'est qu'un pair recherche son successeur.

Dans ce dernier cas, nous lançons en réalité la fonction trouverSuccesseur. Celle-ci va alors déterminer si l'on est le prédécesseur du pair entrant et donc que le pair entrant aura pour successeur notre successeur actuel, pour cela 2 cas de figure:
-Le premier étant un cas normal, notre hash < hash de notre successeur, ainsi on regarde :

-Si notre hash est supérieur au hash donné, dans ce cas on ne s'en préoccupe pas et retransmet directement au successeur.
-Si c'est le cas mais que celui-ci est aussi supérieur à notre hash successeur, on retransmet également.
-Si c'est le cas mais que le hash donné est inférieur au hash suivant, alors on envoi le Hash et l'IP de notre successeur au pair entrant, et ce dernier devient notre successeur.

Cependant il y a des cas exceptionnels. En effet, les hash peuvent être représenté selon un cercle, ainsi un hash 1 peut être le successeur d'un hash 58. Ainsi pour gérer ce cas, il faut dans la condition de décision, y ajouter une partie. Celle-ci fonctionne ainsi:
-Si notre hash >au Hash suivant (fin du cercle), nous regardons le Hash donné, et si celui-ci est supérieur à notre hash mais inférieur à notre successeur, c'est que notre successeur actuel est celui du pair entrant, ainsi il devient notre successeur, et le sien est notre ancien successeur.
- Si notre hash >au Hash suivant (fin du cercle), et que le hash est inférieur à notre hash et celui de notre successeur, on reproduit le résultat du précédent cas.
si ce n'est pas le
-Enfin dans ce cas, si le hash donné > au hash de notre successeur et hash donné>à notre hash, dans ce cas on ne s'en préoccupe pas et retransmet directement au successeur.

Cependant, le système de socket que nous utilisons, nous permet de ne pas avoir 2 fois le même port. Pour cela, nous utilisons les adresses ip, puisqu'elles sont uniques cela nous permet de nous en servir de manière unique du moment que nous les fermons, donc avec un simple split nous pouvons récupérer un port unique, et puisque nous transmettons les adresses IP entre pairs, ces derniers peuvent donc accéder au bon port.

b) Implémentation en cours

Actuellement, nous sommes en train d'améliorer la gestion de nos tables de routage. En effet, pour l'instant nous ne retournons au monitor que le hash et l'IP de notre successeur. Nous travaillons donc sur cette fonctionnalité afin qu'elle puisse retourner le même résutlat que la commande "brout h". C'est à dire, une table de routage complète du réseau P2P. De plus, savons que la taille du serveur welcome et serveur de hash est fixé à 100, nos table comporte donc 7 élements, l'un à 2^0, un à 2^1 ... et le dernier à 2^6 (car $2^6<100<2^7$). Nous aurions pu changer cette taille, mais ne savant pas si l'on pouvait la récupérer à partir de notre classe pair nous l'avons laissé fixe à 100 . Si on avait pu la récupérer on aurait pu créer un classe pair adaptable en fonction de la taille.

Nicolas BLIN - Bryan AUDIC - Corentin GOINARD - Simon ROUSSEAU

INFO 2 - Groupe 1

III - Ajout d'une extension

Concernant l'ajout d'une extension, nous n'avons malheureusement pas eu le temps de la réalisée. En effet, nous disposons pour l'instant d'une version de la classe pair capable d'accéder à un réseau, d'enregistrer et mettre à jour son successeur et son prédécesseur, et de retourner au Monitor son successeur. Cette version a été testée sur 9 machines sans rencontrer de problème.

Par ailleurs, nous étions en train d'améliorer la gestion de la table de routage afin que la commande "rout h" puisse donner le même résultat que la commande "brout h" dans le terminal. Cependant, cette dernière n'est totalement fonctionnelle (test sur 2 machines)

IV - Difficultés rencontrées

Au cours de ce projet, nous avons rencontrés de nombreux et divers problèmes de l'ordre technique mais aussi logique.

Effectivement, nous avons perdu beaucoup de temps au départ lors de la conception et modélisation. Il nous a fallu nous mettre d'accord sur différents points tel que la mise-à-jour du successeur, l'envoie d'un message. Chaque membre du groupe avait une idée bien précise du protocole, par conséquent, nous devions régulièrement discuter de la démarche a suivre.

Par ailleurs, ils nous a été compliqué de nous répartir le travail. En effet, github est un outil nouveau pour nous et nous ne connaissions pas toutes les fonctionnalités dont nous disposions pour mettre notre travail en commun. Cependant, nous sommes conscient que si l'outil avait été bien maîtrisé, il nous aurait permis de gagner beaucoup de temps.

Enfin, nous ne disposions pas toujours de salle informatique afin de déployer notre protocole sur plusieurs machines, et il était impossible de tester notre programme chez nous, ce qui a ralenti notre développement.

V - Conclusion

En somme, au cours de ce projet nous avons conçus un protocole de communication Pair à Pair fonctionnel. En effet, la version actuelle de notre classe Pair permet de faire fonctionner un réseau P2P et a été testée avec un réseau regroupant 9 machines. Cependant, différents points reste à améliorer comme une meilleur gestion de nos tables de routage, ainsi que l'ajout d'une extension à notre projet.

Nous pouvons tout de même retenir de ce projet, l'utilisation des Thread et Socket via le langage Java, ainsi que le fonctionnement d'un protocole Chord correspondant à un modèle Pair à Pair.