Rapport de Projet

Forum de Discussion

RULLIER Noemie et MENORET Clément
24 novembre 2013

Table des matières

1 Introduction

Ce rapport récapitule l'intégralité du travail effectué pour le projet de Middleware. Il aborde de façon générale, l'architecture du projet, le fonctionnement de l'application ainsi que les choix effectués. Ce document traitera de plus de nos réfléxions sur les différentes variantes possibles pour la réalisation de ce système de discussions instantannées. En considérant une nouvelle organisation capable de passer à l'échelle et en détaillant ces caractéristiques.

2 Architecture de notre application

Notre application est un forum de discussion. Un client qui se connecte peut choisir un ou plusieurs sujets et discuter avec tous les autres clients aussi connectés à ce sujet. Les clients peuvent aussi ajouter et supprimer des sujets à volonté.

2.1 Technologie utilisée

Afin de réaliser ce projet nous avons utilisé RMI (Remote Method Invocation) qui est une interface de programmation pour Java. Elle permet d'appeler, d'exécuter et de renvoyer le résultat d'une méthode distante accessible par le réseau.

2.2 Organisation de notre application

Notre application est composée d'un serveur *ServerForum.java*. Ce serveur est chargé de sauvegarder les clients *Client.java* connectés. Il doit de plus conserver une table de correspondance entre le titre du sujet et le serveur gérant ce sujet *SubjectProvider.java*. Chaque SubjectProvider possède un sujet de discussion *SubjectDiscussion.java*.

2.3 Fonctionnement de notre application

La première étape consiste à lancer le serveur, si celui-ci n'est pas en état de marche le client ne pourra pas lancer notre application. Le serveur se lance à partir de la méthode *main* de *MainServer.java*.

Chaque client peut ensuite lancer l'application via la méthode *main* de *MainClient.java*. L'application se connecte au serveur, puis demande au client d'entrer son pseudo. C'est grâce à ce pseudo, qu'on pourra identifier chaque client, et deux personnes ne pourront pas avoir le même pseudo ce qui permettra d'éviter les confusions. Lorsque le client propose un pseudo, celui-ci est soumis au serveur qui valide ou non sa disponibilité.

Le client peut ensuite choisir un ou plusieurs sujets de discussion auxquels il souhaite participer (il peut cependant ouvrir une seule fenêtre pour chaque sujet). Lors de la sélection d'un sujet, le *SubjectProvider* correspondant est renvoyé par le serveur.

La communication se fera ensuite seulement entre le client et le *SubjectProvider*. Chaque sujet contient la liste de ses clients et dès lors qu'un client envoie un message, celui-ci est envoyé à tous les clients abonnés au sujet. Lorsqu'un client quitte un sujet de discussion, le client est supprimé de la liste des inscrits à ce sujet. De même lorsqu'il quitte l'application.

Nous avons intégré deux fonctionnalités qui sont l'ajout et la supression de sujet.

- Chaque client peut créer un nouveau sujet. Il émet sa requête au serveur afin de savoir si le sujet existe déjà (chaque sujet étant identifié par son nom), si il n'existe pas le sujet est ajouté et toutes les interfaces des clients connectés sont mise à jour.
- De même un client peut supprimer n'importe quel sujet, il n'est pas obligé d'etre le créateur. Si des clients sont en cours de discussion sur le sujet, ils sont avertis que le sujet sera supprimé (tous les messages qu'ils pourront envoyer ensuite ne fonctionneront pas même si la fenêtre n'est pas fermée automatiquement). Encore une fois la liste des sujets disponible est mise à jour automatiquement.

3 Structuration pair-à-pair

Une structure pair à pair est proche d'une structure client-serveur mais où chaque client est aussi un serveur, il n'y a donc plus besoin de serveur central lors de l'échange de message. En faisant jouer au serveur le rôle d'entremetteur, nous sommes dans un système pair à pair centralisé. Lorsqu'un client souhaite rejoindre un sujet de discussion, il accède au serveur qui le mettra en contact (en lui renvoyant un ou plusieurs pointeurs distants) avec d'autres clients. Par la suite, les informations circuleront bien directement entre les deux clients.

C'est une solution fragile car si le point central (ici le serveur) tombe, tout le réseau s'effondre. Avec cette nouvelle organisation, il y aura différents points importants à traiter :

3.1 Les points théoriques à résoudre

Si notre système est totalement pair-à-pair, les nouveaux pairs souhaitant s'intégrer au réseau ne pourront plus se connecter facilement à un serveur central déjà connu. Les réseaux pair-à-pair sont par nature dynamiques et on ne pourra pas déterminer automatiquement un point d'entrée sur ce réseau.

3.2 Les points techniques à résoudre

Il faudra prendre en compte de multiples connexions simultannées au niveau des "clients" devenus "pairs" dans cette nouvelle solution. Nous devrons prendre en compte le côté asynchrone des connexions, il sera plus difficile d'assurer la cohérence au niveau des messages échangés entre pairs.

3.3 Solution proposée

Un nouveau pair souhaitant participer aux échanges de messages devra au minimum connaître un point d'entrée sur le réseau. Il serait donc plus pratique de s'organiser en deux niveaux de pairs avec des super-pairs dont l'IP et le port ne changent jamais, que l'on pourrait lister et dont les listes seraient déjà connues des futurs pairs dont la connectivité au réseau pourrait être plus incertaine (connection au réseau pair-à-pair dynamique, à la demande). Une telle architecture nous permettra de répartir la charge plus facilement entre les pairs.

4 Tolérance aux pannes

Actuellement, le serveur central stocke les données dans la RAM uniquement (les sujets de discussions, les clients abonnés, ...). S'il tombe en panne ou est simplement redémarré, toutes les données sont perdues. Nous pouvons donc mettre plusieurs serveurs à la place d'un seul.

4.1 La vivacité

Dans le cas actuel où toutes les données sont centralisées sur le serveur central, si celui-ci n'est plus accessible, c'est tout le système qui ne peut plus fonctionner. Avec plusieurs serveurs répliqués, le système pourra continuer de fonctionner, sans doûte avec des performances dégradées, jusqu'à ce que le dernier serveur fonctionnel ne tombe. Plus il y aura de serveurs, plus la probabilité que le système ne réponde plus du tout diminue grandement car on multiplie la probabilité qu'un serveur tombe en panne à un instant t (< 1) par le nombre de serveurs.

4.2 La convergence

Dans le cas où nous utilisons plusieurs serveurs, une partie importante du système reposera sur la capacité de chaque serveur à répliquer et donc fusionner les modifications de son état avec les nouvelles modifications reçues par les autres serveurs. Nous devrons donc mettre en place un méchanisme à base de numéro de version sur nos données pour s'assurer que les différents états des serveurs seront équivalents lorsque les transferts d'état sont terminés.

4.3 La cohérence des données affichées et des listes d'abonnés

Le méchanisme de versionning que l'on vient d'aborder nous permettrait alors de garantir les caractéristiques d'une eventual consistency. Quand tous les serveurs ont convergés suite aux modifications apportées sur l'état d'un de ces serveurs, la cohérence est assurée.

5 Historique des messages

Dans le contexte de clients qui s'échangent des messages directement de l'un à l'autre en pair-à-pair, l'historique des messages devra être stocké localement chez chaque pair. Chaque pair mettra à jour son propre historique des messages enregistrés dans un fichier. L'historique permettra de conserver les anciens messages échangés lors de précédentes sessions ou avant un crash et permet ainsi aux nouveaux clients d'être informés des précédents messages.

Lorsque'un nouveau pair rejoint la discussion, il est mis au courant des derniers messages reçus par ces voisins et de l'ordre dans lequel ils les ont reçus. Cependant, pour ne pas surcharger le réseau avec le transfert de tout les historiques voisins en même temps, le nouveau pair ne recevra qu'un tout petit sous ensemble des derniers messages. À la demande de l'utilisateur les messages les plus anciens pourront être fournis ultérieurement.

Il faudra limiter le nombre de messages conservés dans le fichier d'historique, par nombre de message et/ou par durée maximale de conservation. Sur un plan technique, on peut limiter la taille en octet du fichier. Dans tous les cas, lorsque la limite est atteinte, on devra opérer à la fois l'ajout du dernier message dans l'historique et la suppression du plus ancien pour toujours maintenir l'historique à jour.

6 Lien entre les solutions proposées

Quelque soit l'architecture choisie, on remarque dans un premier temps que la persistance des données est indispensable dans une optique de tolérance aux pannes et de convergence. En effet, nous voulons assurer à l'utilisateur que quoi qu'il arrive, il sera au courant de ce qui a été dit dans les discussions auxquelles il est abonné et qu'il pourra avoir une représentation de l'état d'un sujet de discussion identique à ce que d'autres utilisateurs verront.

Ensuite, il ne faut jamais oublier que la persistance théorique suppose un espace de stockage illimité, mais dans la réalité, tout client, serveur, pair ne dispose que d'une quantité finie d'espace de stockage, que ce soit en RAM ou sur les disques. Notre solution de persistance doit donc toujours être implémentée en considérant une limite maximale de stockage qui soit un compromis entre la persistance absolue des données et l'espace de stockage qu'on peut - et qu'on veut - lui alouer.

7 Conclusion

Ce projet nous a permis de découvrir la technologie RMI et d'apprendre à l'utiliser. Nous nous sommes, au début, confrontés aux problèmes de compréhension sur le fonctionnement des classes *Serializable* et *Remote*. Une fois ce problème compris, nous avons pu avancer dans notre projet sans trop d'obstacles.

Nous avons ensuite pu réfléchir aux différents problèmes soulevés par la mise en place d'un système pair-à-pair et en discuter au sein de ce rapport. Cette rélexion nous a amené à réflechir à toutes sortes de problèmes comme la persistence des données, la tolérance aux fautes, les aspects de vivacité du système et de cohérence des données.