Rapport Projet Données Réparties : Hagimule

Lécuyer Simon Guillaume Sablayrolles

14 janvier 2025

Résumé

Le projet données réparties a pour objectif de permettre à un client de télécharger en parallèle plusieurs fichiers depuis différentes sources. Ce rapport présente le fonctionnement du projet, sa structure, un manuel d'utilisation et les problèmes rencontrés.

Table des matières

1 Introduction

Le projet données réparties vise à optimiser le partage et le téléchargement de fichiers en utilisant une architecture décentralisée. Inspiré par des modèles peer-to-peer, il permet à plusieurs clients de collaborer pour fragmenter et partager les ressources de manière efficace. Ce projet s'inscrit dans une démarche d'apprentissage sur les systèmes distribués et les problématiques associées, telles que la coordination entre acteurs et la gestion des données.

2 Exemple d'utilisation du programme Tête de Mule

2.1 Lancement de l'Annuaire

1. Lancez l'annuaire :

```
java DiaryImpl
```

2. Précisez l'adresse de l'annuaire :

```
{*Votre adresse dans le réseau*}
```

L'annuaire est un composant central permettant de coordonner les clients et d'assurer le suivi des fichiers disponibles.

2.2 Lancement d'un Client

1. Lancez un client avec la commande suivante :

```
java TeteDeMule
```

2. Renseignez les informations demandées :

```
{*Le nom du diary*}
{*Le port du serveur*}
```

3. Choisissez l'option voulue parmi celles proposées par le programme :

```
1. Voir les fichiers disponibles
2. Télécharger un fichier
3. Ajouter un dossier à la liste des fichiers disponibles
4. Supprimer un fichier disponible au téléchargement
5. Quitter
```

Chaque action est expliquée dans le manuel utilisateur fourni avec le programme.

3 Rappel et Vérification des objectifs

- Un client peut télécharger en parallèle plusieurs fichiers depuis différentes sources : objectif atteint grâce à l'implémentation d'un algorithme de téléchargement fragmenté.
- Annuaire :
  - Enregistrer les clients et les fichiers qu'ils possèdent : fonction implémentée avec succès via une base de données interne.
  - Donner la liste des clients possédant un fichier : opérationnelle, avec des temps de réponse optimisés.
- Client :
  - Enregistrer les fichiers qu'il possède et télécharge : implémenté avec des tests concluants. L'intégrité des fichiers étant bien préservée par le programme.
  - Permettre le téléchargement fragmenté des fichiers : réalisé via une gestion des threads.
  - Télécharger un fichier depuis les clients : fonctionnalité validée par des simulations sur un réseau local.

4 Tests et Performances

Nous avons testé le programme sur le réseaux de l'ENSEEIHT, avec des machines du réseau et en passant par eduroam. La parallèlisation du téléchargement n'est pas le 'bottleneck' en terme de performance de notre méthode mais bien la recomposition des fichiers à postériori. Cependant, nous avons quand même remarqué une amélioration des performances en passant par plusieurs utilisateurs et en parallélisant le téléchargement, notamment sur les gros fichiers. De même, la compression a permis une amélioration significative des performance sur le gros fichiers au détriment de la performance sur les très petits fichiers.

5 Problèmes rencontrés

5.1 Délimitation des rôles des différents acteurs

La délimitation des rôles entre les différents acteurs (clients, annuaire) a été un défi. Il a fallu s'assurer que chaque acteur avait des responsabilités claires et définies pour éviter les conflits et les redondances. Une documentation claire a été rédigée pour clarifier les interfaces.

5.2 Recomposition des fichiers après parallélisation

La gestion des fragments nécessitait une synchronisation précise pour recomposer les fichiers sans erreurs. Des tests unitaires ont permis de corriger plusieurs anomalies. Cependant, cela reste la plus grande perte de temps lors d'un téléchargement. Et nous n'avons pas réussi à trouver de solution évidente au problème car la parallélisation fait arriver les fichiers dans un ordre aléatoire et en des temps qui peuvent beaucoup varier.

5.3 Implantation du démon

Nous avons initialement pensé à l'initialiser au début comme le Diary, mais nous n'arrivions pas à envoyer les données demandées. Nous avons donc décidé de le développer correctement dès le début avec des flux I/O pour gérer le threading. Cela a amélioré la robustesse de la communication entre clients.

5.4 Gestion du Downloader et Démon

Notre premier choix a été de mettre l'instance du démon dans une méthode la regroupant avec le thread du Downloader. Cependant cela empêchait le démon de fonctionner une fois que l'utilisateur avait téléchargé un fichier : le démon se stoppait et ne figurait plus comme actif dans le diary.

5.5 Interface

Nous avons souhaité à l'origine implémenter l'intégralité de l'interface dans le fichier principal `TetedeMule.java`. Cependant, cela nous a empêché de lancer plusieurs téléchargements avec la même instance de l'application car nous avons rencontré des problèmes avec la gestion des threads. Nous avons donc trouvé la solution de déplacer la majorité de l'interface dans le downloader.

5.6 Pare-feu

En testant notre projet avec nos machines personnelles sur le réseau de l'école, nous avons remarqué que le téléchargement est impossible dans un sens. Les pare-feu du réseau semblent empêcher de telles connexions. Il reste cependant possible de télécharger des fichiers disponibles sur les machines de l'école vers nos machines personnelles.

6 Pistes d'amélioration

Avec la structure du projet et les technologies utilisées nous avons isolé trois pistes d'amélioration princiaples :

- La comrpession : pour le moment nous utilisons java.util.zip qui nous permet d'obtenir des réductions de temps sur les fichiers volumineux mais n'est pas optimales. Le but étant de se rapprocher du C pour utiliser le moins de mémoire, nous pourrions utiliser LZ4 pour obtenir une meilleure compression et plus performante.
- La reconstruction du fichier : la reconstruction du fichier est ce qui prend le plus de temps car nous téléchargeons chaque fragment indépendamment puis nous reconstituons le fichier. Pour cela nous utiliserions l'api nio avec nmap permettant d'écrire dans un seul fichier et éviter ainsi la reconstitution des différents fragments.
- La résistance à la deconnexion d'un utilisateur, nous n'avons pas envisagé qu'un utilisateur possédant le fichier se déconnecte pendant le télchargement de celui-ci. Cela pourrait être intéressant pour renforcer la résilience de notre programme.

7 Conclusion

Ce projet nous a permis de mieux comprendre le fonctionnement du téléchargement parallèle, des sockets et RMI. Ayant eu du mal à comprendre comment faire fonctionner le démon, nous avons perdu un peu de temps mais avons compris comment le faire fonctionner et pu avancer par la suite. L'application permet bien le téléchargement de différents fragments entre plusieurs PC de l'école, mais nous n'avons pas testé avec des machines virtuelles à l'étranger pour évaluer l'augmentation potentielle du temps. Ceci est une de nos pistes d'amélioration.

- Guillaume : J'ai travaillé essentiellement sur le Daemon et le Downloader pour l'envoi et la recomposition de fichiers. Je ne comprenais pas comment utiliser les streams avec les sockets pour transmettre la bonne information et donc après avoir échangé avec des camarades, j'ai compris que ce n'était pas forcément un fichier à envoyer mais que cela pouvait être un objet, ici `DataSend`, avec différentes informations. Cette piste nous a permis de terminer le téléchargement et l'envoi de données de façon multi-threadée par la suite.

- Simon : Quant à moi, j'ai plus travaillé sur le Diary et `TetedeMule`, en particulier la gestion des fichiers et la coordination entre les différents modules. J'ai également proposé l'interface principal de l'application.