Rapport de Projet Données Réparties PODP 2

Mathieu Campan, Besson Germain, Simon Hautesserres
Groupe L1

Département Sciences du Numérique
2022-2023

Table des matières

Introduction

Ce rapport décrit nos choix de conception et les algorithmes que nous avons implémenté au cours de l'étape PODP 2.

1 Ajout apportés :

1.1 Propagation de l'objet :

La propagation de l'objet a été implanté dans la suite logique de notre code. En effet, nous propagions déjà un compteur de notifications pour que le client connaisse le nombre de modifications en attente qu'il doit consulter. Il nous a ainsi suffit de rajouter l'objet en paramètre pour compléter la mise à jour.
Dans cette étape nous avons principalement traité les problèmes de blocage/interblocage qui sont expliqués en 2 Choix de conception.

1.2 Tests :

Nous avons également mis à jour nos tests créés durant le projet PDR : ClientNormal et ClientLazy sont deux classes représentant des applications qui font des demandes d'accès en écriture et lecture de façon continue (le Lazy ayant un délai d'attente avec le verrou). Nous avons ainsi rajouté un abonnement et désabonnement sporadique. Pour cela nous avons également codé une classe ObservateurClientNormalLazy qui affiche les mises à jour de l'objet par abonnement dans un fichier test_abonne.txt.

Pour le lancer, il suffit d'executer :
sh lancement_test.sh

Ce script lance un certain nombre (modifiable dans le fichier) de clients normaux et lazy qui feront des accès concurrentiels sur l'objet Irc. Le résultat de ces accès et des abonnements est visible dans le fichier test.txt et test_abonne.txt.

2 Choix de conception :

2.1 Unlock :

Nous avons rendu l'appel à unlock non-bloquant grâce à un thread Slave, afin de s'assurer que le client lâchant le verrou ne soit pas bloqué lors de la propagation de l'objet mis à jour aux abonnés. En effet, il est possible d'avoir de très nombreux abonnés et l'attente de la propagation serait trop importante, alors que le client n'est pas censé s'en préoccuper.

2.2 Blocages :

Nous avons rendu les fonctions subscribe et unsubscribe du ServerObject synchronized afin d'éviter que plusieurs processus modifient la liste des abonnés en même temps : problèmes d'accès concurrentiel. De plus, nous avons également utilisé un thread Slave lors de la propagation de l'objet mis à jour aux abonnés dans le ServerObject. Ceci pour permettre au Server de revenir traiter les demandes d'accès à l'objet sans perdre de temps à attendre la mise à jour des abonnés, ce qui pourrait donner lieu à un interblocage.