Rapport d'objets dupliqués

Maxime Arthaud & Korantin Auguste

22 janvier 2014

1 Introduction

À première vue, le sujet nous paraissait relativement intéressant : un système gérant des objets partagés mène à des problèmes complexes de concurrence. De plus, ce système pourrait être utilisé dans des cas concrets (comme par exemple NFS).

2 Étape 1

Le code de base est venu naturellement à partir du squelette fourni.

Nous avons rapidement abouti à une version fonctionnelle, mais après développement d'un « fuzzer » pour torturer un peu notre système, des bugs (notamment des $NullPointerException$) ainsi que des deadlocks sont apparus. Même si la plupart ont pu être résolus assez rapidement, certains apparaissaient dans des cas très complexes.

Pour nous aider à trouver la raison des deadlocks, nous avons mis en place un système de log qui nous permet de savoir tout ce qui se passe, aussi bien sur le serveur que sur les clients.

Le problème le plus difficile a été le cas de la figure 11 des slides (deux $lock\_write$ en parallèle, alors qu'un des deux clients a déjà un RL). Pour résoudre cela, nous faisons en sorte que l'appel de $Client.lock\_write$ n'empêche pas les $invalidates\_reader$.

À la suite de ce problème, un autre est survenu : il est possible de recevoir un $lock\_*$ et un $invalidate\_*$ de la part du serveur à peu près en même temps, et donc le $invalidate\_*$ peut passer avant. On a mis une condition dans les $invalidate\_*$ qui, dans le cas où on n'a pas de lock, attend qu'un $lock\_*$ se termine bien.

3 Étape 2

Nous avons commencé par envisager l'utilisation de classes $Proxy$ pour créer une classe dynamiquement qui serait capable d'exposer les méthodes originales de manières transparentes, mais avons fini par abandonner devant certains problèmes de java (impossible de créer une classe dynamiquement qui hériterait d'une autre classe).

Nous nous sommes donc rabattu sur l'utilisation de l'API « reflect », pour faire de l'introspection et générer du code java, ce qui a l'avantage d'être simple, mais assez malsain.

4 Étape 3

La phase 3 a consisté principalement à rajouter une méthode pour sérialiser des $SharedObject$, ce qui n'a pas posé de difficulté particulière (il suffit d'écrire seulement l'identifiant du $SharedObject$), même si des tests ont pu montrer qu'il y avait quelques corrections annexes à effectuer (penser à enregistrer l'objet dans la $HashMap$ des clients au moment de la désérialisation).

5 Conclusion

Le projet n'a pas été toujours évident, principalement à cause de la gestion de la concurrence dans tous les cas tordus qui peuvent arriver en montant en charge. Au final, nous pensons toutefois avoir quelque chose de très robuste !

Le reste n'a pas vraiment posé problème. Ah si : on aime pas le Java, mais encore sur ce projet, ça allait (pas comme le JEE, qui est une horreur à coté d'un vrai framework web..).