Systèmes Concurrents
Évaluation du travail réalisé par le groupe A

Pierre Jeanjean - Sacha Liguori

Jeudi 15 Décembre 2016

1 Partie technique

1.1 testWriteAndTryRead

Le but de ce test était simplement d'ajouter des tuples avec take(), puis de les récupérer avec tryRead(), en vérifiant que les tuples obtenus correspondent bien à ceux ajoutés.
Test validé

1.2 testTake

Ce test consistait à write() un tuple, puis d'appeler successivement take() et tryRead(). On vérifie que le take() retourne bien le bon tuple, et le supprimer (tryRead() retourne null).
Test validé

1.3 testRead

Comme précédemment, en remplaçant take() par read(). Le tryRead() doit alors également fonctionner.
Test validé

1.4 testTakeLock

Simple test vérifiant que la méthode take() est bien bloquante.
Test validé

1.5 testReadLock

Simple test vérifiant que la méthode read() est bien bloquante.
Test validé

1.6 testTryTake

Test vérifiant que tryTake() retourne bien null si le tuple n'existe pas, retourne le bon tuple sinon, et supprime le tuple retourné.

Test validé

1.7 testWriteWithEditions

Test plus intéressant, vérifiant à l'aide d'éditions que les tuples sont copiés en profondeur, que ce soit lors de leur ajout ou de leur lecture/retrait.

Test validé

1.8 testReadAll

Test vérifiant que readAll() retourne bien tous les tuples nécessaires, ou une collection vide si aucun ne correspond.

Test validé

1.9 testTakeAll

Test similaire au précédent, mais pour takeAll(). Vérifie en outre que les tuples correspondant sont bien supprimés.

Test validé

1.10 testReadTakeSynchronized

Test vérifiant que l'écriture d'un tuple par un second Thread durant l'attente des méthodes read() et take() est bien prise en compte, et réveille potentiellement le premier Thread. Ce test a une durée maximale pour s'exécuter (afin de vérifier la réactivité).

Test validé

1.11 testEvents

Test relativement complexe, vérifiant que les événements soient traités de façon correcte :

— Différence entre les modes IMMEDIATE et FUTURE
— Suppression du tuple par TAKE et pas par READ
— Les événements ne sont déclenchés qu'une fois
— Les événements sont déclenchés de façon cohérente

Ce test a une durée maximale pour s'exécuter (afin de vérifier la réactivité).

Test validé

1.12 Qualité du code

La qualité du code est bonne, il est bien commenté et documenté.

L'utilisation des Lambdas Expressions propres à Java 8 rend la lecture du code simplifiée et réduit le nombre de lignes de code.

Seule remarque : la définition de la classe privée CallbackRef n'a rien à faire entre deux méthodes publiques de CentralizedLinda...

2 Synthèse

Concernant la correction et la complétude, nous n'avons rien à dire : nous n'avons détecté aucune bug ni spécification non respectée.

Le travail effectué est également pertinent, et nous avons particulièrement appréciée la gestion des événements (qui influe très peu sur la performance du reste des fonctionnalités). Nous avons cependant quelques doutes sur la nécessité d'effectuer l'opération read() en zone critique (une gestion transactionnelle serait probablement plus adaptée et efficace. De plus, actuellement, chaque appel à la méthode write() notifie tous les Threads en attente de read() ou de take() pour qu'elles tentent de s'exécuter une nouvelle fois. Comme chacune de ces méthodes doit s'exécuter en zone critique, on imagine facilement de gros ralentissements dans des scénarii mettant en scène un très grand nombre de Threads. Il serait donc préférable de ne notifier que les Threads ayant un intérêt à l'être.

Pour ce qui est de la cohérence, notre seule remarque concerne la gestion des Threads, décrite précédemment : la gestion des événements est actuellement très efficace, et les Threads pourraient facilement être traités de façon similaire, donc pourquoi ne le sont-ils pas ?

Pour résumer, voici quelques pistes d'amélioration :

- Déplacer la définition de la classe CallbackRef pour une meilleure visibilité
- Repenser la concurrence : tous les appels à read() ne nécessitent pas de s'effectuer en exclusion mutuelle, et une gestion transactionnelle semble plus adaptée pour obtenir de meilleures performances
- La gestion des Threads en attente mériterait d'être améliorée, en s'inspirant par exemple de la gestion des événements, encore une fois pour améliorer la performance