Rapport de projet de données réparties

Fainsin Laurent - 2SN M2
Guillotin Damien - 2SN M2

Organisation du projet

Le projet est écrit en Java (17) et ses dépendances sont gérées par gradle. Les fonctionnalités développées sont regroupées dans des packages et leur détails seront explicités plus bas. Pour effectuer des tests nous utilisation la librairie JUnit5 et pour vérifier que nos tests sont efficaces nous utilisons Jacoco (pour obtenir le code-coverage).

Pour lancer les test JUnit5:

```
gradle tasks test
```

Pour lancer Jacoco:

```
gradle tasks jacocoTestReport
```

Pour ouvrir le rapport Jacoco:

```
firefox build/reports/jacoco/test/html/index.html
```

Organisation des packages

linda

Ce package contient les interfaces et les classes nécéssaires à l'ensemble du projet. Nous n'y avons pas touchées, celles-ci étant fournis par le sujet.

linda.shm

Ce package contient l'implémentation `CentralizedLinda` de l'interface `Linda`. Ainsi, ont été écrites les primitives:
`write`, `take`, `tryTake`, `takeAll`, `read`, `tryRead`, `readAll`, `eventRegister`.

`CentralizedLinda` permet la création de l'espace partagé de données typées Linda centralisé sur la machine sur laquelle le programme est lancé.

De nombreux tests ont été effectués sur cette implémentation via `linda.test.CentralizedLindaTests`. Le code coverage atteint presque 100%.

linda.server

Pour partager l'implémentation précédente entre plusieurs machines/processus, nous pouvons l'enregistrer dans un registre RMI et créer les interfaces associées pour permettre aux clients/servers de communiquer.

Ainsi nous avons créé l'interface `LindaRemote`, reprenant les méthodes de `Linda`. De même nous avons créé la classe `LindaServer` implémentant `LindaRemote` et dont le but est de publier dans un registre RMI une instance `CentralizedLinda`. `LindaClient` vient simplement chercher une instance `Linda` dans le registre RMI et passe l'ensemble de ses actions à cette instance.

Pour les tests nous avons simplement repris ceux de `CentralizedLinda` mais nous les avons adaptés pour que ceux-ci fonctionne avec RMI.

linda.eratosthene

Cette application est constituée d'un serveur responsable de distribuer les tuples à tester, et de clients responsables d'effectuer des calculs sur ces tuples. Ainsi `Server` instancie un `LindaServer` de la partie précédente et initialise le tuple-space avec des tuples formés d'un entier (l'entier dont ont veut tester la primalité) et de deux strings qui indiquent l'état de test de cet entier. Chaque `Client` se contente simplement de se connecter au serveur et de vérifier la primalité d'un tuple, une fois son calcul terminé, il place le résultat dans le tuple-space. Cette dynamique persiste tant qu'il existe encore un entier non testé dans le tuple-space.

Pour les tests nous comparons simplement les entiers retournés avec des entiers dont nous sommes sûrs qu'ils soient premiers.

linda.search.basic

Cette application fonctionne directement grâce au code `Server` et `Client` écrit précédemment. Ainsi, comme demandé, nous avons ajouté les fonctionnalités :

- avoir plusieurs activités de recherche (qui traitent concurremment la même requête).
- avoir plusieurs activités manager (qui déposent plusieurs requêtes).
- l'utilisation avec le linda serveur : le(s) manager(s) et le(s) chercheurs sont chacun un client distinct.
- le démarrage dynamique et arbitraire de chercheurs : avant le dépôt ou alors que des chercheurs ont déjà commencé. La terminaison du manager devient un point délicat.
- l'arrêt arbitraire de chercheurs ; le manager doit réaliser qu'il n'y en a plus aucun.
- le retrait de la recherche par le manager après un certain délai (par exemple 5 secondes) alors que les chercheurs n'ont pas fini. Dans ce cas, il est important de laisser le système dans un état propre, tel que les chercheurs puissent commencer une nouvelle recherche si un nouveau manager en dépose une.

Pour les tests nous lançons simplements plusieurs intances de `Manager` et `Searcher` et nous comparons avec des résultats connus.