Projet Données Réparties

Planification

Paul Anaclet

Guohao Dai

Théo Desprats

2A SN

2022

1 Corrections apportées à la version originale

Nous n'avons pas apporté de corrections particulières à la version précédente car suffisamment fonctionnelle.

2 Améliorations envisagées pour les performances

2.1 Structures de données

Afin des faciliter la localisation d'un `Tuple` dans l'espace de stockage, nous pensons mettre en place une `HashTable` de la forme suivante :

```
HashTable<Integer, ArrayList<Tuple>> tuples ;
```

Cette `HashTable` aurait comme clé un entier correspondant à la taille des `Tuples` (c.à.d le nombre d'éléments dans le `Tuple` récupérable grâce à `size()`) contenus dans l'`ArrayList` associée. Cela permettra une séparation des tuples et pourra potentiellement rendre leur accès plus rapide.

Nous avons choisi `HashTable` car implémentée de manière « thread-safe » mais nous ne sommes pas sûr d'être en accord avec les objectifs pédagogiques du projet. Si ce n'est pas le cas, nous utiliserons une `HashMap` et effectuerons la synchronisation nécessaire.

2.2 Parallélisation

Chaque serveur Linda aura un pool fixe de `LindaThreads` traduits par une nouvelle classe étendant `Thread`. Nous transformerons les primitives Linda (`take()`, `write()`...) en tâches qui seront soumises à ce pool par le biais des `ExecutorServices`.

La synchronisation des accès aux `ArrayLists` de `Tuples` et aux `Callbacks` se fera sur le modèle Lecteurs/Rédacteurs équitable (avec un système de tours) : plusieurs Lecteurs seront autorisés à accéder en lecture à la structure de données et un Rédacteur n'est autorisé qu'à écrire que s'il est seul.

2.3 Cache

Pour mettre en place un cache de tuples côté client qui réduira la charge des serveurs, nous comptons modifier la classe `LindaClient` en lui ajoutant un attribut :

```
HashTable<Integer, ArrayList<Tuple>> tuplesCache ;
```

Cette `HashTable` fonctionnera comme celle décrite en section 2.1 et sera interrogée par le client avant chaque appel au serveur Linda. Nous hésitons aussi à seulement utiliser une `ArrayList<Tuple>` comme le cache sera de taille relativement faible par rapport à l'espace des serveurs.

En cas de suppression d'un `Tuple` dans l'espace, le serveur devra propager l'information aux clients: il faudra donc qu'il ait une liste des Clients connectés sur lesquels il pourra appeler une méthode distante invalidant le `Tuple` supprimé.

2.4 Répartition

La répartition de l'espace de tuples entre les serveurs se fera comme indiquée dans le sujet : un client dépose et recherche des tuples d'abord sur le serveur auquel il est connecté, puis, en cas d'échec, le serveur propagera la requête aux autres.

Pour cela, chaque serveur devra posséder une liste contenant les serveurs « voisins », grâce à laquelle ils pourront propager les primitives en appelant leurs méthodes.

3 Mesures des performances

3.3 Paramètres et critères

Nos mesures des performances s'effectueront selon les critères suivants :

- temps d'exécution : temps écoulé entre l'appel et le retour d'une primitive depuis le client, temps de recherche dans l'espace de tuples côté serveur...

- utilisation de l'espace mémoire : coût de l'exécution, utilisation processeur...

- fiabilité / résilience : taux d'erreurs, comportement en cas de coupure d'un ou plusieurs serveurs, d'impossibilité de communication serveur/serveur ou serveur/client...

- exploitation des serveurs : répartition de la charge globale, des tuples…

3.1 Adaptation de la plateforme Linda

La mesure des performances selon les critères précédemment définis implique l'instrumentalisation de `LindaServer` : il faut que l'interface permette l'obtention de données liées à l'état actuel du serveur pour que l'on puisse les exploiter plus tard :

- état du serveur : pourcentage de charge, temps de vie, temps d'inactivité...

- état de l'espace des tuples : nombre de tuples stockés, taille maximale de l'espace…

- état sur les échanges : nombre d'écriture/lecture, volumes des échanges...

Il serait possible d'implémenter un « getter » par type d'information disponible ou bien une seule méthode `getData()` retournant un seul objet (JSON ?) contenant toute les données et dont le format serait clairement spécifié et définit. Nous n'avons pour l'instant pas choisi de réponse quant à cette question.

Nous considérons également la mise en place d'un mode verbeux / log des serveurs, activable au lancement afin d'aider le débuggage en cas de problèmes (utilisation de bibliothèques spécialisées) ainsi que d'un ou plusieurs serveurs Linda de backup pour rendre la plateforme un peu plus résistante à la panne.

3.2 Outils

Afin de mesurer les performances et exploiter les informations de Linda, nous allons développer plusieurs outils proposant plusieurs fonctions :

- Application de test : écrite en Java, permettra, selon les arguments au lancement, de simuler des accès de nombre, de tailles, de types, et de temps plus ou moins aléatoires afin d'observer le comportement de la plateforme.

- CLI : écrit en Java, proposera des commandes paramétrées pour exécuter des scénarios, consulter et modifier l'état et les informations d'un serveur Linda, mesurer les performances d'une application exécutée et possiblement générer des courbes (évolution du nombre de tuples/callbacks, du volume des échanges en fonction du temps etc...). Il faudra donc effectuer une connexion au serveur, utiliser les nouvelles primitives d'instrumentalisation de Linda avec de l'horodatage ('get' sur les différentes informations) et exploiter les données (affichage texte, output un fichier dot...)