Systèmes Concurrents:
Projet : partie 1.2

Groupe B : Thibault MEUNIER Matthieu PERRIER

6 Janvier 2017

Sommaire

Introduction

Ce rapport présente l'ensemble des tests effectués sur l'implémentation du serveur linda faites pas le groupe A, ainsi qu'une présentation de l'interpréteur.

1 Interpreteur

Dans l'archive fournie se trouve un interpreteur de commande dans le package "outils.interpretor". Pour le lancer, executer simplement RunShell. L'objectif a ete de fournir un outil puissant offrant toutes les fonctionnalités de Linda et permettant de simuler facilement des comportements complexes. Ainsi nous nous sommes orientes vers un interpreteur travaillant sur des processus qui interragissent avec un unique Linda. L'utilisateur commence tout d'abord par generer les processus via la commande dediee 'create', puis lui affecte une tache a realiser. Chaque processus etant independant, il n'y a pas de probleme de blocage lors d'une attente. Un processus en attente sera detecte par l'interpreteur et toute interaction avec celui-ci sera refusee. Il est egalement possible de jouer des scenarios par l'intermediaire de la commande 'read'. Il est important de noter que ces derniers sont executes dans l'environnement courant. Il est donc possible d'en voir les resultats et de continuer a interagir avec eux. On notera egalement la presence d'un operateur de parallelisation '||' permettant de lancer plusieurs commandes en parallele. Les processus etant deja parallelises, son interet reside essentiellement dans le lancement de scenarios. Pour plus d'informations, voici le resultat de la commande 'help':

```
These are common Linda commands used in various situations:
create <name>:    create a process with given name
list: print all processes
read <file>: execute commands from file within the same environment.
debug: print the tuple space
exit:    close the current shell and all associated process
<process name> <action>: execute an action on process
     write <tuple>:    write the given tuple
     take <tuple>:        take a matching tuple
     read <tuple>:        read a matching tuple
     tryTake <tuple>:    try to take a matching tuple
     tryRead <tuple>:    try to read a matching tuple
     takeAll <tuple>:    take all matching tuples
     eventRegister <read|take> <immediate|future> <tuple>: register event as described
     terminate:         terminate the process
You can parallelized request with the || operator.
Lines starting with "// " are not processed.
```

2 Tests Unitaires

Chacunes des primitives linda a été soumise à une batterie de tests écrits en JUnit. Un fichier de test a été créé pour chaque primitive testée.

Les points suivants ont été testés : -Ecriture/ lecture correcte des tuples demandés -Renvoit d'une NullPointerException lorsque le tuple écrit ou lu est null -Blocage/Déblocage lorsque nécessaire de processus sur les primitives Read et Take

L'ensemble de ces test a été passé sans dévoiler aucune erreur. On peut donc dire que la spécification du serveur est respectée par cette implémentation.

3 Tests de performance

Vous trouverez les résultats de tous les test suivants en annexe.

3.1 Impact de la position

Ce test permet de déterminer l'impact de la position du tuple dans le serveur sur son temps d'accès (Read). Il apparaît clairement que le temps d'accès au premier tuple écrit sur le serveur est nettement inférieur a celui du dernier tuple. Cependant, il n'est pas constant, ce qui aurait pu être attendu étant donné que les tuples sont stockés dans une liste.

3.2 Performance Mémoire

Sans surprise, on observe que le temps d'accès à un tuple au milieu du serveur dépend du nombre de tuple stockés par celui ci. L'utilisation d'ArrayList permet cependant d'avoir un temps d'accès qui ne dépend pas linéairement du nombre de tuples.

3.3 Capacité mémoire

Ce test consiste à écrire autant de tuple possible sur le serveur avant d'obtenir une OutOfMemoryException. Les résultats sont très variable suivant la machine sur lequel tourne le serveur, mais on observe que celui fonctionne sans erreur jusqu'à saturation de la RAM de la machine l'hébergent.

3.4 Performance face à de nombreux processus

Ici, on cherche à déterminer le temps d'accès moyen à un tuple par de nopmbreux processus simultanéments. On observe que le temps d'accès moyen est a peu près linéaire au temps nombre de processus. On note également que ce temps est fortement impacté par le nombre total de tuples sur le serveur.

3.5 Comparaison Read et Take

On observe que la primitive nécessite moins de temps que la primitive take. Ce phénomène s'accentue lorsque le nombre de tuple sur le serveur est important. Ceci s'explique par la nécessité de supprimer les tuple du serveur dans le second cas.

3.6 Temps pour vider le serveur

Ce test consiste en l'exécution d'un takeAll qui correspond à l'intégralité des tuples stockés sur le serveur. On observe que le temps d'exécution augmente très vite et fortement lorsque le nombre de tuples a récupérer est grand. Ceci s'explique par la recopie des données avant de les renvoyer à l'utilisateur. Ce choix n'est pas le plus judicieux et il aurait pu suffire d'utiliser les éléments existant (contrairement au ReadAll). Ceci est surement le seul point négatif que le peut relever sur les performances de ce serveur.

4 Tests Concrets

4.1 Semaphore

L'implantation d'un semaphore a ete faite par l'intermediaire d'un linda dedie. On y ecrit un tuple a chaque appel V et on en prend un a chaque P. La prise de tuple etant bloquante, on obtient la caracteristique attendue pour un semaphore. Cependant un semaphore qui se veut une structure legere de synchronisation et l'implantation par l'intermediaire de l'API Linda est extremement couteuse. De plus, cet outil est a la base de nombreuses autres structures plus complexe et on peut craindre une deterioration exponentielle des performances.

4.2 Moniteur

L'implantation des moniteurs s'est faite en deux parties : la gestion du moniteur et celle des conditions. La premiere partie consiste en la mise en place d'une exclusion mutuelle sur les operations. L'utilisation d'un semaphore comme mutex fut alors choisie. La seconde resida dans l'implantation du systemes des conditions et du signalement. Lorsqu'une condition est enregistre, elle possede un semaphore dediee. Celui-ci est alimente par la methode signal et les utilisateurs sont mis en attente avec la methode await.

4.3 Applications classiques

Pour illustrer les capacites des outils que nous avons cree precedemment, nous avons choisi de reprendre les applications que nous avons vu en tp. Ainsi on trouvera plusieurs implantations du probleme des philosophes et du probleme des lecteurs et redacteurs. Le respect des versions interface Java dans nos implantations ont permis un portage facilite de ces applications. Elles nous ont aussi permis de verifier la pertinence de nos tests en offrant un environnement

beaucoup plus riche et peuplee de donnees aleatoires et importante. Les implantations du probleme des philosophes se concentrent sur les semaphores, tandis que les lecteurs redacteurs utilisent des moniteurs.

4.4 Eratosthene

Lors de la creation d'exemple de fonctionnement de l'API Linda, il nous etait demande de presenter l'algorithme d'eratosthene sous une forme parallelise. Afin de simplifier la tache et d'attester du bon fonctionnement des operations utlisees, nous avons commencer par implanter une version sequentielle de cette algorithme gerant la crible au travers d'un espace de tuple. A l'initialisation, l'espace est rempli d'entier de 2 a une limite fixee par l'utilisateur. Ensuite, l'utlisateur lance l'algorithme et peut recuperer la crible resultat. Pour paralleliser cet algorithme, l'idee a ete de repartir l'algorithme en plusieurs zones a traiter independamment. On reduit ainsi les temps de traitements. On pourrait ameliorer l'efficacite en appliquant un traitement systematique pour les multiples de nombres entiers petits qui constitue l'essentiel des traitements. En choisissant de retirer tous les nombres pairs, on gagne significativement en temps de calcul. On remarquera que cette version est plus lente qu'une version sans parallelisation et n'utilisant pas Linda en Java pur.

5 Conclusion

Annexe : Résultats des tests de performance

Impact de la position

Nb Tuples	Tps 1er (ms)	Tps dernier (ms)	Tps milieu (ms)
50000	15	17	4
100000	14	21	4
150000	8	12	5
200000	7	13	6
250000	13	20	9
300000	8	15	8
350000	9	17	11
400000	16	25	13
450000	13	21	14
500000	14	26	15

Performance Mémoire

```
Temps pour 50000 tuples : 9 ms
Temps pour 100000 tuples : 9 ms
Temps pour 150000 tuples : 4 ms
Temps pour 200000 tuples : 5 ms
Temps pour 250000 tuples : 6 ms
Temps pour 300000 tuples : 6 ms
Temps pour 350000 tuples : 7 ms
Temps pour 400000 tuples : 8 ms
Temps pour 450000 tuples : 9 ms
Temps pour 500000 tuples : 10 ms
Temps pour 550000 tuples : 11 ms
Temps pour 600000 tuples : 11 ms
Temps pour 650000 tuples : 13 ms
Temps pour 700000 tuples : 13 ms
Temps pour 750000 tuples : 14 ms
Temps pour 800000 tuples : 16 ms
Temps pour 850000 tuples : 16 ms
Temps pour 900000 tuples : 17 ms
Temps pour 950000 tuples : 18 ms
Temps pour 1000000 tuples : 18 ms
```

Performance face à de nombreux processus

nProcess \ nTuples	5000	10000	15000	20000	25000	30000	35000	40000	55000	50000
100	19	9	6	14	22	15	17	58	40	62
200	4	25	42	37	75	40	85	53	70	88
300	18	28	57	52	87	55	60	130	110	168
400	12	49	47	40	144	81	145	117	159	175
500	28	29	107	95	145	73	117	139	174	233
600	48	45	57	118	130	83	159	199	235	285
700	48	38	44	134	104	183	216	276	285	327
800	45	52	115	100	87	189	200	252	320	396
900	37	100	111	75	248	208	316	357	295	438
1000	52	99	173	215	151	188	385	361	406	541

Comparaison Read et Take

Nb Tuples	Tps Read (ms)	Tps Take (ms)
50000	9	16
100000	6	12
150000	4	10
200000	5	14
250000	6	17
300000	6	15
350000	8	24
400000	8	25
450000	9	25
500000	10	24

Temps pour vider le serveur

Nb Tuples	Tps (ms)
100	140
200	482
300	366
400	501
500	765
600	1047
700	1368
800	1753
900	2222
1000	2753