TP Bases de données parallèles

Objectif

Réaliser une version parallèle de l'algorithme de jointure relationnel par hachage, en utilisant les threads Java. Comprendre les problèmes de synchronisation entre des taches concurrentes en utilisant des données partagées, et les résoudre en contrôlant l'asynchronisme.

Définition du problème

Pour simplifier les aspects jointure relationnelle (gestion de tuples), on considère deux ensembles d'entiers à joindre :

R = {1 3 5 7 9 … 99}, card(R)=50
S = { 1 2 5 6 9 10 … 100 }, card(S)=60

La jointure de R et S se réduit alors à calculer T = R ∩ S = {1 5 9 … 99}. En utilisant 10 threads, le problème est de paralléliser le calcul de T de sorte que chaque thread i calcule $T_i = R_i \cap S_i$.

Solution 1 (initialisation séquentielle)

Un thread producteur dispose de R et S, et crée 10 sous-ensembles (correspondant à des paquets) R_i et 10 sous-ensembles S_i, avec la même fonction de hachage (modulo 10). Le thread producteur réalise donc :

```
pour chaque r dans R
     i = r mod 10
     insérer r dans Ri
pour chaque s dans S
     i = r mod 10
     insérer s dans Si
créér 10 threads consommateurs
```

Chaque thread consommateur i calcule $T_i = R_i \cap S_i$. Enfin, un thread résultat calcule $T = T_1 \cup T_2 \cup T_3 \ldots \cup T_{10}$.
A faire : implémenter, tester et démontrer cette solution.

Solution 2 (initialisation parallèle)

La production des paquets est faite en parallèle par 10 threads producteurs. On suppose que R et S sont déjà partitionnés sur 10 threads comme suit :
Le thread 1 dispose de R_1 = {1 3 5 7 9} et de S_1 = {100 99 96 95 92 91}, le thread 2 de R_2 = {11 13 15 17 19} et de S_2 = {90 89 86 85 82 81}, etc.

A faire : modifier la solution précédente pour paralléliser la production des paquets. Attention à synchroniser les threads (en utilisant les méthodes wait, notify, …).