TME DataStore : KVStore

Dans un contexte data science, nous étudions un cas d'analyse de données dynamiques subissant des modifications. Une étape préliminaire à l'analyse consiste à stocker les données et gérer les modifications (e.g. update) que demandent les applications. Pour les différentes raisons présentées en cours (passage à l'échelle, disponibilité, ...) les données ne sont pas stockées dans un SGBD relationnel mais dans un système NOSQL.

Le 1er exercice vise à appréhender les différentes façons de structurer les données stockées dans un système NOSQL. Dans le cas de données dynamiques subissant des modifications, il est souhaitable d'adapter la structure des données en fonctions des modifications que demandent les applications.

Exercice 1. NOSQL KVStore

On considère une application devant gérer un catalogue de produits dans une base NOSQL KVStore. Les Produits sont stockés à l'aide des paires (clé, valeur) suivantes :

La composante principale (*i.e.* majeure) est formée de /Produit/id/N où N est le numéro de produit.

Les composantes secondaires (*i.e.* mineures) sont /nom et /prix

/Produit/id/1/ - /nom	livre A
/Produit/id/1/ - /prix	10
/Produit/id/2/ - /nom	film B
/Produit/id/2/ - /prix	10

L'application effectue deux transactions E et L :

E (id1, id2, S) augmente de S le prix des produits identifiés par id1 et id2

L (id1, id2) lit le prix des produits identifiés par id1 et id2.

Les deux produits n'ayant pas la même composante principale, il n'est pas possible de demander à KVStore d'exécuter de manière transactionnelle une liste d'opérations d'écritures (*cf.* la méthode execute(List<Operation>) vue en TP).

Question 1) Exécuter le programme *InitMagasinGlobal.java* qui affecte un prix identique aux produits id1 et id2. Puis exécuter le programme *ExecutionSansTransaction.java* mettant en évidence le phénomène de lecture incohérente lorsque **E** et **L** ne sont pas des transactions. Expliquer les méthodes suivantes : tacheLecture(int duree), tacheEcriture(int duree). Expliquer le déroulement du programme. En particulier, expliquer pourquoi des lectures incohérentes peuvent se produire quand **L** s'exécute.

Question 2) Proposer une solution qui évite les lectures incohérentes. On veut observer que la lecture des prix des produits id1 et id2 aboutit à lire deux prix identiques. Pour cela, on modifie la structure de la base :

/Produit/id/1/ - /nom	livre A
/Produit/id/2/ - /nom	film B
/Produit/prix - /id/1	**10**
/Produit/prix - /id/2	**10**

Initialiser le store en exécutant le programme *InitMagasinLocal.java*.

Expliquer les modifications à apporter aux méthodes tacheLecture et tacheEcriture. Implanter la solution dans le programme *LectureEcritureLocale.java*. Définir **L** et **E** comme des transactions accédant aux clés **/Produit/prix - /id/1** et **/Produit/prix - /id/2** partageant la même composante principale**.**

Question 3) Un inconvénient de la solution proposée à la question précédente est que toutes les occurrences de **L** et **E** sont traitées sur la même partition contenant la composante principale **/Produit/prix**., quel que soit le produit. Pour réduire cet inconvénient, on souhaite distribuer les opérations **L** sur plusieurs partitions. Proposer une solution stockant le prix de manière redondante telle que :

Une opération **E** modifie, dans la même partition, les prix associés aux clés **/Produit/prix - /id/1** et **/Produit/prix - /id/2** puis les nouveau prix sont répercutés sur les clés **/Produit/id/1/ - /prix** et **/Produit/id/2/ - /prix**.

Les **L** sont distribuées autant que possible sur plusieurs partitions en lisant les clés **/Produit/id/1/ - /prix** et **/Produit/id/2/ - /prix**. La solution doit éviter les lectures incohérentes. Pour cela, on peut considérer qu'on connait le numéro d'arrivée de chaque opération **E** (*cf.* numE). Si nécessaire, compléter la base avec de nouvelles clés. Par exemple, une clé **/Produit/id/1 - /derniere** pour stocker le numéro de la dernière transaction ayant modifié le prix dont la clé est **/Produit/id/1 - /prix**.

Utiliser *InitMagasin3.java* pour initialiser le store avec les prix redondants. Détailler les avantages et inconvénients de la solution que vous proposez. Implanter la solution proposée dans le programme *LectureEcritureGlobale.java*.

Exercice 2. DynamoDB

Pour cet exercice, répondre aux questions en français (on ne demande **pas** d'implantation).

On considère une base NOSQL instanciée à l'aide d'un service similaire à AWS DynamoDB. La qualité du service est définie comme suit. Quand on crée une table, il faut indiquer (i.e., acheter) une capacité d'accès que le service DynamoDB s'engage à fournir. Cette capacité s'exprime par deux valeurs : le nombre de lectures par secondes (appelé DebitTotalLecture ou $\mathbf{D_L}$) et le nombre d'écritures par secondes (appelé DebitTotalEcriture ou $\mathbf{D_E}$).

On suppose qu'une table est partitionnée en **N** partitions et que la capacité d'accès est répartie de manière égale sur chaque partition. En conséquence sur chaque partition, la capacité est limitée à :

- Nombre de lectures par secondes par partition : $\mathbf{D_{PL}} = D_L / N$
- Nombre d'écritures par secondes par partition : $\mathbf{D_{PE}} = D_E / N$

De plus, l'espace de stockage associé à une partition est limité à 10Go. Une application peut créer 20 tables au maximum.

L'application étudiée stocke dans la table Avis, les avis que des utilisateurs émettent sur des produits. La clé de la table Avis est *numProduit*. Les items de la table Avis ayant la même valeur pour *numProduit* sont triés selon l'attribut *date*, dans l'ordre croissant. Les autres attributs sont le *nom* du rédacteur de l'avis, la *note* (de 1 à 5) et le *commentaire.*

Les données sont partitionnées sur l'attribut *numProduit*.

Les deux traitements applicatifs étudiés sont :

- **L**(p) : lire les 5 avis les plus récents concernant le produit numéro *p*
- **E**(p, r, n, c) : insérer un nouvel avis sur le produit numéro *p* rédigé par *r* avec la note *n* et le commentaire *c*.

Traiter **L** nécessite une lecture. Traiter **E** nécessite une écriture.

Initialement, on décide d'utiliser 80% de l'espace de stockage sur chaque partition (soit 8Go). La base ayant une taille d'environ 200Go (1million de produits * 200Ko par avis), la table Avis est formée de N = 200/8 = 25 partitions. On sait que l'application a besoin de traiter au maximum 100 écritures par secondes et 1000 lectures par secondes. On fixe D_E=100 et D_L=1000. Sur chaque partition, on a D_{PE} = 100/25 = 4 et D_{PL} = 40.

On constate un problème d'accès lorsqu'un produit très populaire p_1 reçoit beaucoup de nouveaux avis provenant de plusieurs personnes : pendant certaines périodes, l'application a besoin de traiter 10 écritures par secondes concernant le produit numéro p_1. La plupart de ces écritures sont refusées car cela dépasse la capacité D_{PE}=4 de la partition contenant p_1.

1) Proposer une solution pour résoudre ce problème et permettre le traitement de 10 écritures par secondes pour le produit p_1 tout en conservant une valeur de D_E égale à 100 pour la table Avis.

2) On considère que les produits populaires varient dans le temps : un produit populaire peut devenir progressivement de plus en plus populaire jusqu'à recevoir 200 écritures par seconde pendant quelques heures, puis sa popularité disparait, puis autre produit peut devenir populaire. Certains jours aucun produit n'est populaire. On suppose qu'il est possible de modifier la valeur de D_E d'une table mais pas plus d'une fois par jour. On suppose que le coût du service facturé est égal à la somme de l'usage de toutes les tables. Le coût d'usage d'une table est :

Coût = $(D_L + D_E) * h$ avec h étant le nombre d'heures d'existence de la table.

Par exemple, si on crée une table avec DL=1000 et DE=100 et qu'on supprime cette table au bout de 4h, alors l'usage a coûté 4400. Proposer une solution capable de traiter les écritures en tenant compte de la popularité variable des produits tout en essayant de réduire le coût du service. Est-ce que la solution a un impact sur le traitement de L(p) ?