Parcours DataScale
Architecture des Gestionnaires de Données

Examen 16 novembre 2015

Durée : 3h. Tous (vos) documents autorisés.
Soyez précis dans vos réponses, toute réponse non justifiée sera considérée comme fausse.
Utilisez deux feuilles séparées pour les Parties 1 et 2.

PARTIE 1 (14 points) : Architecture et mécanismes internes des Gestionnaires de Données

EXERCICE 1 (5 points) : Transactions

Supposons une exécution parallèle de 4 transactions T_1, T_2, T_3 et T_4 produisant, sans contrôle de concurrence, l'histoire (ou séquence d'opérations) suivante :

1	2	3	4	5	6	7	8	9	10	11	12	13	14
$R_1(a)$	$W_1(a)$	$R_2(b)$	$R_4(d)$	$R_3(c)$	$W_3(c)$	$R_4(a)$	$W_2(d)$	$R_1(c)$	$W_1(b)$	C_1	C_2	C_3	C_4

On utilise la notation usuelle, à savoir $R_i(x)$ (resp. $W_i(x)$) représente une opération de lecture (resp. écriture) par la transaction T_i de l'objet x. C_i représente le commit de la transaction T_i.

Question 1.1 : Cette exécution est-elle sérialisable ? Quelle que soit la réponse, donnez-en la preuve.

Question 1.2 : Quelle est l'histoire produite par un protocole d'estampillage ?

Question 1.3 : Quelle est l'histoire produite par un protocole de certification ?

Question 1.4 : Quelle est l'histoire produite par un protocole de verrouillage à 2 phases classique ?

Question 1.5 : même question que 2.4 mais on considérant cette fois-ci que T_2 s'exécute en degré d'isolation Read Committed.

Pour les questions 2.2 à 2.5, indiquer simplement les actions intéressantes et l'étape (de 1 à 14) à laquelle elles se produisent.

EXERCICE 2 (5 points) : Indexation

Considérons la table `Commandes(IdCommande, IdClient, IdProduit, Date, Quantité)` contenant 1.000.000 de commandes, passées par 10.000 clients, sur 50 produits en magasin, dans des quantités pouvant aller de 1 à 20 maximum. Chaque page disque fait 4KB et peut contenir 50 tuples de `Commande`. On souhaite avoir (1) un accès très rapide à toutes les commandes d'un même client (requête Req1 avec critère `IdClient` = v) et (2) un accès rapide aux commandes relatives à un certain produit commandé dans une certaine quantité (Requête Req2 avec critère `IdProduit` = v1 and `Quantité` = v2).

Question 2.1 : quels types d'index (Plaçant/non plaçant, B-Tree/hachage/bitmap/bloom filter) priviligeriez-vous pour la table `Commandes` et pourquoi ?

Question 2.2 : quelle serait la taille en pages d'un index bitmap sur `IdProduit` et sur `Quantité` ?

Question 2.3 : En supposant les données équi-réparties, combien de tuples seraient résultat d'une requête de type Req1 et Req2 ?

Question 2.4 : En supposant qu'une entrée/sortie aléatoire coûte 5 millisecondes et une entrée/sortie séquentielle 0,2 milliseconde, quel serait le temps approximatif d'exécution des requêtes de type Req1 et Req2 ?

EXERCICE 3 (4 points) : Quiz

La réponse à chacune des questions ci-dessous devra être justifiée avec précision.

Question 3.1 : Soit la jointure de R avec S (avec |R|<|S|) exécutée en Block-Nested Loop et soit M le nombre de pages de RAM allouées pour cette jointure. On réserve 1 page de RAM pour bufferiser l'écriture du résultat. Il reste donc M-1 pages à répartir entre le buffer de lecture de R et celui de S. Vaut-il mieux (1) donner une seule page pour lire R et les M-2 pages restantes pour lire S, (2) faire l'inverse ou (3) répartir équitablement entre les 2.

Question 3.2 : Soit une transaction T distribuée sur 2 sites S1 et S2. Lors de la validation atomique de T, S1 et S2 votent tous les deux READY puis une panne réseau isole S2. Le coordinateur peut-il valider T1 sur S1 ? Que se passe-t-il sur S2 ?

Question 3.3 : La mémoire Flash NAND supporte mal les écritures disques aléatoires. Parmi les techniques d'indexation B-Tree, hachage et bitmap, laquelle vous parait la mieux adaptée à cette contrainte ?

Question 3.4 : En quoi le Database Cracking est-il mieux adapté au Big Data qu'une méthode d'indexation traditionnelle ? Peut-on affirmer que le Database Cracking reste systématiquement meilleur qu'une indexation traditionnelle quel que soit le nombre de requêtes exécutées sur les même données ?

PARTIE 2 (6 points) : Systèmes de gestion de données dans le cloud

EXERCICE 4

Question 4.1 (3 points): Partitionnement statique

Considérons une application de type e-commerce gérant principalement trois collections de données :

- collection de 500.000 clients avec pour chaque client son profil (de l'ordre de 2 K octets par client) ;

- collection de 200.000 articles à vendre classés en 50 catégories, avec pour chaque article une description détaillée, un prix et une quantité en stock (de l'ordre de 5 K octets par article) ;

- collection de 2.500.000 commandes, en moyenne 5 par client, avec un minimum de 1 et un maximum de 100 (de l'ordre de 1K Octet par commande).

Sur cette base de données, on exécute principalement les transactions suivantes : (1) ajouter un client avec son profil ; (2) passer une commande (associe un client, un article et une quantité) ; (3) top100 des articles les plus vendus ; (4) ajouter un article dans une catégorie (avec description, prix et stock).

On veut mettre en place cette application dans le cloud en utilisant un système de gestion de données de type clé-valeur. Pour cela on désire appliquer un modèle de partition statique qui devra assurer l'absence de transactions distribuées pour l'ensemble des transactions décrites ci-dessus.

Proposez un tel mode de partition statique pour chaque collection, en expliquant pour chaque transaction en quoi ce partitionnement garantit l'absence de l'usage de transactions distribuées.

Question 4.2 (1 point) : Transactions multi-clés

Pourquoi un système clé-valeur classique (n'implémentant que des transactions sur un seul couple clé-valeur) ne peut pas être utilisé tel quel dans une application utilisant un modèle de partition dynamique ?

Question 4.3 (1 point) : Migration de bases de données à chaud

Est ce que la migration à chaud complète de bases de données prend plus de temps dans le modèle « shared nothing » que dans le modèle « decouplage storage » ? Expliquer pourquoi.

Question 4.4 (1 point) : Migration de bases de données à chaud dans le système Albatross

Pourquoi n'y a-t-il pas de requêtes en échec (« failed requests ») pour la migration à chaud d'Albatross alors qu'il y en a des centaines si on utilise une stratégie très simple de migration de type Stop & Migrate ?