université PARIS-SACLAY | FACULTÉ DES SCIENCES D'ORSAY

30/10/2019

Programmation Distribuée
Master 1 ISD

Examen

durée 2h00

L'examen comporte 3 exercices sur 2 pages. Le barême sur 20 points est donné à titre indicatif et est proportionnel à la difficulté des exercices. Tous les appareils (téléphones, ordinateurs portables, ...) doivent être rangés. Les documents autorisés sont une feuille A4 personnelle recto verso par support de cours.

1 Transactions SQL et sérialisabilité (6 points)

1. (1.5 point) Répondre par **vrai** ou **faux** aux affirmations suivantes, sans justifier.
   (a) Pour être en conflit, deux actions doivent se trouver dans des transactions distinctes.
   (b) Deux actions sur des objets différents peuvent être en conflit.
   (c) Étant donné deux actions, pour qu'elles soient en conflit il faut que l'une d'elle soit une écriture.
2. (2.5 points) Soit l'ordonnancement d'actions ci-dessous. Dire s'il est sérialisable. Vous justifierez en déssinant le graphe de dépendances. Chaque arrête du graphe sera étiquetée $i \to j$ où i et j sont les numéros des deux actions en conflit (donnés entre parenthèses à côté de chaque action).

T1	T2	T3	T4
r(X) (1)			
	r(Y) (2)		
		r(X) (3)	
	w(Y) (4)		
		r(W) (5)	
		w(Y) (6)	
w(Y) (7)			
			r(W) (8)
			w(X) (9)
r(Z) (10)			
w(Z) (11)			

3. (2 points) On se donne la table SQL suivante :

```
CREATE TABLE R (ID VARCHAR(10) PRIMARY KEY, VALUE INTEGER);

INSERT INTO R VALUES ('W', 1);
INSERT INTO R VALUES ('X', 2);
INSERT INTO R VALUES ('Y', 3);
INSERT INTO R VALUES ('Z', 4);
```

   Donner, pour chacune des transactions abstraites de la question 2. Une séquence d'ordre SQL qui possède la même comportement de lecture et d'écriture. Vous pouvez considérer que les objet `W`, `X`, `Y`, `Z` correspondent aux lignes dont l'`ID` est `'W'`, `'X'`, `'Y'`, `'Z'`. On rappelle qu'une condition `SELECT` accède à la ligne en lecture, `DELETE` accède à la ligne en écriture et `UPDATE` effectue une lecture suivi d'une écriture. Si c'est opérations sélectionnent leur lignes par clé primaire, alors seul la ligne concernée est touchée. Si elles utilisent une autre condition, alors le toute la table est impactée en lecture et les lignes sélectionnées sont impactées en écriture.

2 Multihread en Java (7 points)

On considère l'implémentation minimale d'une file, donné ci-après. Le contenu de la file est représenté par une liste chaînée pour laquelle on garde une référence vers l'arrière de la file (où l'on ajoute) et l'avant de la file (d'où on retire).

```
class LList<T> {
  LList<T> next;
  T value;
  LList() {
    value = null;
    next = null;
  }
}

public class Fifo<T> {

  private LList<T> front;
  private LList<T> back;
  private int size;

  public Fifo() {
    front = null;
    back = null;
    size = 0;
  }

  public synchronized int size() {
    return size;
  }

  public synchronized boolean isEmpty() {
    return front == null;
  }

  public synchronized void add(T t) {
    if (back == null) {
      //avant == null aussi
      back = new LList<>();
      front = back;
    } else {
      LList<T> l = new LList<>();
      back.next = l;
      back = l;
    }
    back.value = t;
    size++;
  }

  public synchronized T remove () {
    if (front == null)
      throw new Exception ("Empty FIFO");

    T t = front.value;
    front = front.next;
    if (front == null)
      back = null;
    size--;
    return t;
  }
}
```

1. (2.5 points) Donner du code Java créant une file de chaînes de caractères (`File<String>`) puis créant et démarrant **deux threads**, l'un ajoutant 20000 chaînes entières `"0"`, `"1"`, ..., `"19999"` dans la file et l'autre lisant ces chaînes tour à tour et les écrivant dans la sortie standard (vous pouvez choisir de mettre les *threads* soit dans des classes séparées soit dans des classes anonymes). Vous utiliserez la classe comme si elle était sûre du point de vu de l'accès concurrent.
2. (2.5 points) On suppose, **uniquement pour cette question**, que toutes les occurrences du mot-clé `synchronized` ont été retirées de du code. Donner un ordonnancement possible d'instructions exécutées par les deux threads de la question 1 qui mettent la file dans un état incohérent (exception levée, cellule de la liste « perdue », l'un des attributs `front` ou `back` valant `null` sans que ce soit le cas pour l'autre, ...).
3. (2 point) Compléter le code de la méthode `forEach` ci-dessous. Cette dernière prend en argument un objet `q` de type `Fifo<E>` et un objet `c` de type `Consumer<E>`. Ce dernier est un objet possédant une méthode `void accept(E e)` qui effectue une action sur `e` sans renvoyer de résultat. La méthode `forEach` doit appeler `c.accept(`v`)` pour chaque élément v de la file `q`. Vous n'avez pas le droit de modifier le code de la classe `Fifo<E>`, la méthode `forEach` se situe à l'extérieur de cette classe et cette méthode doit être sûre du point de vue de l'accès concurrent.

```
public class Utils {
  public void forEach(Fifo<E> q, Consumer<E> c) {
    //à compléter
  }
}
```

3 Map/Reduce et Spark (7 points)

1. (6 points) Pour chacune des transformations demandées ci-dessous, donner en pseudo-code un ensemble de fonctions `map` et `reduce` permettant de les calculer. Vous décrirez précisement les entrées et sorties et indiquerez dans quel ordre exécuter les paires de fonctions maps et reduces s'il y en

a plusieurs. Pour simplifier l'écriture, quel que soit le pseudo-langage utilisé, vous avez à votre disposition une fonction `write(key, value)` qui envoie la paire $(\texttt{key}, \texttt{value})$ à la phase d'après (dans le shuffle pour le `map`, et dans la sortie pour le `reduce`).

(a) (2 points)

entrée une séquence de couples (n, l) où n est un entier représentant un classement ATP et l une liste de chaînes de caractères représentant des noms de joueurs ayant atteint ce classement.

sortie une séquence de couples (j, n) où j est un nom de joueur et n le meilleur classement qu'il ait jamais eu (le numéro le plus petit).

(b) (4 points)

entrée une séquence de paires (n, s) où s est une chaîne de caractères représentant une ligne d'un texte et n est la position de cette ligne dans le fichier original.

sortie une séquence de couples (c, m) indiquant pour chaque caractère la longueur de la plus grande répétition de ce caractère dans le fichier d'origine.

2. (1 point) On considère le fragment de code Scala suivant, inspiré de l'exemple WordCount :

```
1  val textFile = sc.textFile("hdfs://...")
2  val wordList = textFile.map(line => line.split("\\W"))
3  val mappedWordList = wordList.map(word => (word.toLowerCase(), 1))
4  val reducedList = mappedWordList.reduceByKey((x,y) => x+y)
5  reducedList.saveAsTextFile("hdfs://...");
```

Le compilateur indique une erreur de typage sur l'appel `word.toLowerCase()` indiquant que le type `Array[String]` ne possède pas cette méthode. Indiquer d'ou vient l'erreur et comment la corriger.