2. Übungszettel

Vorlesung Transaktionale Systeme

Abgabe *am Do. 29/04/2010 in der Übung*

H. Schweppe, J. Broß

Aufgabe 1 (Isolationseigenschaften) 3 Punkte

Betrachten Sie folgende Historien und beschreiben Sie informell die Isolationseigenschaften. Welche Konflikte treten auf? Welche Phänomene (lost update, dirty read, …) treten auf?

a) h1 = r1[x] r2[x] w1[y] r3[z] w3[z] r2[z] r2[y] r3[y] w3[y] c1 c2 c3

b) h2 = r1[x] r3[x] r2[x] w1[x] w3[x] w2[x] a3 w1[y] r2[y] w2[z] r1[z] w1[z] c2 a1

Aufgabe 2 (Serialisierbarkeit) 4 Punkte

a) Zu zeigen: die Konflikt-Äquivalenz von Historien ist eine Äquivalenzrelation (also insbesondere transitiv).

b) Ist der Serialisierungsgraph einer Historie transitiv, d.h. gilt
Ti -> Tj , Tj -> Tk => Ti -> Tk ? Beweis oder Gegenbeispiel!

c) Ist die *serielle* Historie, die es zu einer *serialisierbaren* Historie gibt, eindeutig?

Projekt:

Implementierung eines Lock Managers:

Schreiben Sie eine LockManager in Java, der folgendes Interface implementiert (die vollständige InterfaceDefinition kann von der ProjektHomepage zur Vorlesung heruntergeladen werden):

```
public interface ILockManager {
      public boolean lock(int xid, String strData, LockType lockType)
            throws DeadlockException;

      public boolean unlockAll(int xid);
}
```

Ein LockManager speichert Informationen darüber, welche Transaktion (hier durch eine numerische TransaktionsID repräsentiert) Sperren auf welche Objekte hält, bzw.

auf solche wartet. Objekte könnten Datensätze oder ganze Seiten sein, in unserem o.g. Interface werden sie einfach durch einen eindeutigen String referenziert.

Es können Lese oder Schreibsperren angefordert werden, wobei jeweils nur eine Transaktion eine Schreibsperre auf ein Objekt halten darf, während mehrere Transaktion gleichzeitig Lesesperren auf ein Objekt haben können. Die Sperren einer Transaktion werden alle gleichzeitig freigegeben, d.h. dieses Interface würde sich z.B. für rigoroses 2PL eignen.

Über die Speicherung von Sperren hinaus ist es die Aufgabe eines LockManagers, Verklemmungen zu erkennen. Eine einfache Lösung dafür wären Timeouts, die aber problematisch sind. Hier soll daher Deadlock-Erkennung durch Wartegraphen realisiert werden. In einem Wartegraphen sind die Transaktionen die Knoten und gerichtete Kanten zeigen von Transaktion, die eine Sperre anfordern auf solche, die gerade eine unverträgliche Sperre halten. Wenn in diesem Graphen ein Zyklus auftritt, handelt es sich um eine Verklemmung und eine der Transaktionen im Zyklus muss abgebrochen werden. Zu beachten: es sollte möglich sein, dass Transaktionen ihre Sperren „upgraden“, d.h. aus einer Lesesperre wird eine Schreibsperre.

a) Erarbeiten Sie bis zum 29.04.2010 ein Konzept zur Umsetzung des Lock Managers. Schauen Sie sich dazu insbesondere die Funktionalität der Klassen im `java.util.concurrent` Paket der Java-Klassen-Bibliothek an.

b) Implementieren Sie bis zum 06.05.2010 den Lock Manager. Geben Sie ein jar-File ab, das Ihre Implementierung des Lock Managers enthält (ggf. Hilfsklassen). Achten Sie darauf den Quellcode mit einzufügen.

c) Achten Sie darauf, dass die zur Verfügung gestellten JUNIT-Tests mit Ihrer Implementierung erfolgreich durchlaufen.