Synchronisation in Datenbanksystemen – in a nutshell

1. Modell für nebenläufige Transaktionen und Korrektheitskriterium

Transaktionsmodell: Folgen von Lese und Schreiboperationen abgeschlossen durch c=commit. :

$$TA\ \mathbf{j} = r_j(x)\ r_j(y)\ w_j(y)\ r_j(z)\ w_j(x)\ w_j(s)\ w_j(z)\ c_j$$

die atomar ("Alles oder nichts") ausgeführt wird. Sie soll gegen Einflüsse nebenläufiger TA geschützt sein (Isolationseigenschaft). Es werden nur erfolgreiche Transaktionen betrachtet.

Die Operationen nebenläufig ausgeführter Transaktionen (TA) werden im Allgemeinen verzahnt ausgeführt z.B.

r1 (x) r2(y) r2(z) w2(y) r2(x) r1(y) w2(x) w1(y) r1(z) r2(s) c2 w1(x) c1

Der **Scheduler** des Datenbanksystems (Teil der **Transaktionssteuerung**) soll für eine korrekte Reihenfolge der Operationen sorgen.

Korrekt ist die Reihenfolge per Definition, wenn der **Effekt auf die DB dem Effekt einer beliebigen seriellen Ausführung** der nebenläufigen Transaktionen ("eine nach der anderen") **entspricht**. Die Ausführungsfolge heißt dann serialisierbar (**SERIALIZABLE**).

Gesucht sind **Verfahren** *(Synchronisationsverfahren)*, die dieses Korrektheitskriterium oder ein abgeschwächtes Kriterium erfüllen.

Abgeschwächte Kriterien sind:
READ COMMITTED: Ein TA liest nur Werte, die durch den erfolgreichen Abschluss einer TA festgeschrieben wurden.

REPEATABLE READ: Lesen innerhalb einer TA ist wiederholbar. Der Zustand eines von TA gelesenen Objekts darf während der Dauer von TA nicht von anderen Transaktionen verändert werden.

REPEATABLE READ ist stärker als READ COMMITTED:

r1 (x) BOT_2 r2 (x) w2 (x) c2 r1 (x) c1 (BOT= Beginn der TA)

T1 liest nur festgeschriebene Werte, sieht aber zwei Zustände von x.

2. Synchronisationsverfahren

2.1 Sperrbasierte Verfahren

Zweiphasig Sperren (Two Phase locking, 2PL)
Grundprinzipien:
a) Jedes Objekt, das gelesen oder geschrieben wird, muss vor der Operation gesperrt werden. Im Allgemeinen gibt es **kompatible Lesesperren** (unterschiedliche

Transaktionen dürfen dasselbe Objekt nebenläufig lesen) und **exklusive Schreibsperren.**
b) "**No lock after unlock**" – sobald eine Sperre freigegeben wurde, darf keine weitere angefordert werden.
Zweiphasig sperren garantiert Serialisierbarkeit.

Bespiel mit Sperranforderungen:

R_lock1(x) r1 (x) r1 (x) c1

BOT_2 R_lock2(x) .....r2 (x) W_lock(x) w2 (x) c2
T2 wartet auf Sperrfreigabe

Die Freigabe erfolgt fast immer **strikt**, d.h. zum Commit-Zeitpunkt.

Ein weiteres sperrbasiertes Verfahren (**Pre-Claiming**, Vorabanforderung aller Sperren) vermeidet Verklemmungen, erfordert aber eine genaue Kenntnis aller benötigten Objekte und wird in DBS nicht verwendet.

2.2 Optimistische Verfahren

Optimistische Verfahren haben drei Phasen:
Lesephase: Kopieren aller Daten in einen privaten Speicher und Durchführen der Operationen.
Validierungphase: Prüfen, ob es einen Konflikt mit einer nebenläufigen TA gegeben hat.
Schreibphase: Schreiben der veränderten Daten in die Datenbank.

Varianten:
Backward-oriented Optimistic CC – validieren gegen die Transaktionen, die seit Beginn der validierenden TA erfolgreich beendet wurden (commit).

Forward-oriented Optimistic CC - validieren gegen die noch aktiven Transaktionen.

BOCC wird mit Änderungszeitstempel realisiert: ein Objekt x besitzt den Zeitstempel des Commit der letzten schreibenden Transaktion. Damit muss eine validierende TA prüfen, ob alle gelesenen Objekte den gleichen Zeitstempel besitzen, wie in der Lesephase.

Die Historien sind serialisierbar.

Optimistische Synchronisation kann zu einem TA-Abbruch führen, währen bei zweiphasigem Sperren lediglich eine Wartesituation entsteht:

T1: r(x) _____Validierung um x zu schreiben, Fehlschlag
T2: r(x) ___ Validierung _Schreibphase.

2PL:
T1: r(x)___ w(x) _____c1

T2: [r(x)] …….. r(x) w(x) c2 ([…] Wartesituation, x für T1 gesperrt)

2.3 Mehrversionen Verfahren

Es gibt mehre Versionen jedes Objekts. (Implementierung z.B. durch Rekonstruktion von Tupeln aus der Protokollierungsdatei).
Für die meisten Verfahren benötigt ein Objekt als Zeitstempel den Commit-Zeitpunkt der Transaktion, die die letzte Änderung durchgeführt hat.

Lesetransaktionen: Verzicht auf Lesesperren
Eine Lesetransaktion R erhält von allen gelesenen Objekten die Version, die zum Zeitpunkt des Beginns von R aktuell war. Das Verfahren ist in Kombination mit 2PL serialisierbar. R wird so "in die Vergangenheit verschoben " (durch Lesen der geeigneten älteren, aber konsistenten Versionen), dass R keinen Konflikt mit späteren Änderungstransaktionen aufweist.

R: r(x0) _____r(y0)

T1 r(x0) r(y0)_____w(y1) c1

Die Eigenschaft READ ONLY muss dem Transaktionsmanager vorab bekannt sein!
Lese / Schreib-Transaktionen können z.B. das 2PL-Protokoll benutzen.

2.3.1 Consistent Read

Ziel: Isolationslevel "READ COMMITTED" erreichen.
Verfahren: Jede TA liest die letzte bestätigte ("committed") Version jedes Objekts. *Schreibsperren* verhindern das gleichzeitige erstellen von neuen Versionen. *Keine Lesesperre erforderlich* (!)

T2 r2(x0) r2(y1) r2(x1) c2 (liest nur "committed")
R3(x0) R3(y0) R3(x0) (R/O Transaktion)

T1 w1(x1) w1(y1) c1
T4 r4(x0) [w2(x2)]_____w2(x2) c3 [....] = warten auf Freigabe
Achtung: könnte zu *lost update* führen können! Es gibt keine langen Lesesperren.

Deshalb: Eine TA, die ein Objekt x zum Schreiben liest, benötigt als potentieller Änderer von x eine Schreibsperre.

T1 [w1(x1)]_____w1(x1) w1(y1) c1
T4 r4(x0) w2(x2) w2(x2) c3 (Schreibsperre auf x0)

Verfahren garantiert nicht REPEATABLE READ, aber es werden nur bereits festgeschriebene Werte (*committed*) gelesen.
Analog im Beispiel: Will T2 x verändern, muss eine Schreibsperre angefordert werden, damit w2(x2) ausgeführt werden kann.
Lesesperren r2(x0) usw. werden dagegen überhaupt nicht angefordert – es gibt keine!

2.3.2 Snapshot Isolation

Ziel: REPEATABLE READ.

Verfahren:

1) Transaktionen lesen die **Version eines Objekts, die bei *Beginn der TA*** aktuell war ⇒ **keine Lesesperren** benötigt.
2) Die **Schreibmengen von je zwei TA müssen disjunkt sein**.

Sperrbasierte Implementierung

T1 will *x* schreiben:

Wenn x nicht gesperrt
Wenn die Versionsnummer > BOT(T1) dann Abbruch
// es gab jüngeren Schreiber, der Ausgangszustand von x bzgl. T1
// verändert hat.
Sonst sperre x und verändere, halte Sperre bis commit;
Sonst // x gesperrt)
wartet T1 auf Ende der die Sperre haltenden TA T2:
- commit T2 ⇒ abort T1
- abort T2 ⇒ commit T1

Bespiel:

T1: r1(y0) r1(x0) [w1(x1)].....abort

T2: r2(x0) w2(x2) c2 Warten, da T2 x modifiziert, erfolgreich (c2), deshalb abort (T1)

Eine Implementierung mit Validierung analog zu optimistischen Verfahren kommt ohne Sperren aus. Geprüft wird, ob das Objekt x inzwischen verändert wurde ("*First committer wins*")

Beide Implementierungen der Snapshot ISOLATION sind in vielen Fällen sogar serialisierbar, erfüllen also nicht nur REPEATABLE READ.
(Fragen: Warum eigentlich "REPEATABLE READ"?
Warum sind im Allgemeinen mehr als 2 Versionen erforderlich?)

2.3.3 Zweiversionen-Verfahren "2PL2VMVCC"

Es gibt nur zwei Versionen, eine davon aktuell. Beide sind im Normalfall gleich, eine kann aber bei Gewährung einer Schreibsperre zu einer neuen Version gemacht werden. Diese Version ist nicht sichtbar, bevor sie nicht "publiziert" (d.h. zur aktuellen Version gemacht) wurde.

Das Publizieren einer Version, bei der die vorherige Version ihre Gültigkeit verliert, darf erst geschehen, wenn kein Leser mehr auf dieser Version aktiv ist.

Verfahren: zusätzliche Sperre ("certification lock"), die angefordert wird, wenn x zur neuen Version gemacht werden soll. Sie ist inkompatibel zu Lese- und Schreibsperren, damit die neue Version nach endlicher Zeit publiziert werden kann.

Lese- und Schreibsperren sind kompatibel, da Lesen und Schreiben auf getrennten Versionen geschieht. Schreibsperren verhindern, dass zu einer Zeit mehr als eine Version entsteht.

Will man das Verfahren mit Read-Only Transaktionen kombinieren, stößt man auf eine Schwierigkeit: es gibt nur 2 Versionen. Damit erkannt wird, dass nach Erwerb einer *certify*-Sperre kein Leser mehr auf x aktiv ist, muss jede TA, auch reine Leser, eine Lesesperre erwerben, die aber (fast) immer gewährt wird. Einzige Ausnahme: es gibt schon eine *certify*-Sperre auf dem Objekt. Es werden alles in allem nur Leser bevorzugt, da sie nicht wegen einer vorhandenen Schreibsperren auf x warten müssen, sondern lediglich für die Dauer der in der Regel kurzen *certify*-Sperre.