Databases and Information Systems

9. Übungsblatt

(Testatwoche: 15. - 17. Juni 2010)

Lösung

Einführung in Datenbanksysteme

Heinz Schweppe, Katharina Hahn

Aufgabe 1

a) Geben Sie ein Beispiel für eine Ausführungsfolge (Schedule) für 2 Transaktionen an, bei dem im Fall von 2PL-Synchronisation (eine Version!) ein Deadlock auftritt, bei Schnappschuss-Isolation (snapshot isolation) jedoch beide Transaktionen erfolgreich mit commit abgeschlossen werden können.
b) Geben Sie ein Beispiel für das Gegenteil an: 2PL bewirkt eine Wartesituation mit erfolgreichem Abschluss der Transaktionen, dagegen führt Schnappschuss-Isolation zum Abbruch einer Transaktion.
c) Gibt es einen Unterschied in 1), b), wenn man für Schnappschuss-Isolation die optimistische (first comitter wins) bzw. die pessimistische Varianten (Sperren) wählt?
d) Bei 'consistent READ MVCC ' können lost updates auftreten. Geben Sie ein Beispiel an.

Lösung: *2+2+1+1 Punkte*

a) $r_1(x_0)\ r_2(y_0)\ w_1(x_1)\ w_2(y_2)\ r_1(y_0)\ r_2(x_0)\ c_1\ c_2$
$rl_1(x)\ r_1(x)\ rl_2(y)\ r_2(y)\ wl_1(x)\ w_1(x)\ wl_2(y)\ w_2(y)\ rl_1(y)$...wait 1 → 2
$rl_2(x)$ ...wait 2 → 1

b) $c_0\ r_2(y_0)\ r_1(x_0)\ w_1(x_1)\ c_1\ r_2(x_0)\ w_2(x_2)$ → abort!
$rl_2(y)\ r_2(y)\ rl_1(x)\ r_1(x)\ wl_1(x)\ w_1(x)\ c_1$—$ul_1(x)\ ul_1(y)\ rl_2(x)\ r_2(x)\ wl_2(x)\ w_2(x)\ c_2$—$ul_2(all)$

c) Nein: in a) kein w-w-Konflikt,also überhaupt keine Auflösung nötig,
in b) wird der w-w-Konflikt bei „first committer wins“ evtl später erkannt, aber abgebrochen wird in jedem Fall.
d) Bei 'consistent READ MVCC ' können lost updates auftreten. Geben Sie ein Beispiel an.
T1: $r_1(x_0)$ …. $w_1(x_1)$ wait...$w_1(x_1)$ // lost update
T2 $r_2(x_0)$ $w_2(x_2)$ c_2

Aufgabe 2 *1+2 Punkte*

a) Geben Sie eine Ausführungsfolge (Schedule) von Transaktionen an, für die gilt: bei 2PL-Sperren wartet eine Transaktion, wird jedoch mit COMMIT beendet, bei optimistischer Synchronisation wird die Transaktion zurückgesetzt.
b) Ist es möglich, dass 2 Transaktionen bei 2PL-Synchronisation verklemmen, bei optimistischer jedoch beide zum commit kommen? Beispiel oder Beweis.

Lösung:

a)
T1 r(x)_____w(x) _____c
T2 r(y)_____ r(x).....r(x)__w(x)___c

Optimistisch:
T1 r(x)_____r(x)_____val+w(x)
T2 r(y)_____r(x)___w(x)_____val, Negativ, Wert von x hat sich geändert! abort

b)

T1 r(x) r(y) w(x) wait for release of x by T2
T2 r(x) r(y) w(y) wait for release of y by T1

Optimistisch: falls nur geprüft wird, ob die zu schreibenden Objekte nach BOT der validierenden Transaktion geändert wurden, kommen beide zum Commit. Die Historie ist aber nicht serialisierbar.

Andernfalls (Prüfen des gesamten Read-Sets der validierenden TA T auf Disjunktheit zum Write-Set aller TA, die zwischen BOT und Validierung von T erfolgreich abgeschlossen wurden) gilt das nicht. Es existieren nach Voraussetzung zwei gegenläufige Konfliktpaare, z.B. ($r_1(x)$, $w_2(x)$) und ($r_2(y)$, $w_1(y)$). Die Transaktionen sind nebenläufig und eine validiert zuerst, z.B. T2. R(T1) ∩ W(T2) ≠ ∅, also wird T1 abgebrochen. (Formal müsste der Beweis Kreise beliebiger Länge im WF-Graphen berücksichtigen. Das lässt sich – technisch etwas schwieriger – durch Induktion zeigen.)

Aufgabe 3 *(4) Punkte*

Für hierarchische Sperren (mit verschiedenen Größen der Sperrobjekte) waren die Modi R (Leseberechtigung), X (Exklusiv), IR (Lese-Intention) und IX (Intention des exlusiven [Schreib-]zugriffs) eingeführt worden. Erweitern Sie das Verfahren um den Sperrmodus RIX, der *das Objekt im R-Modus sperrt* und *alle Unterknoten mit einer IX-Sperre* versieht. Dieser Modus kann beim Lesen einer Tabelle und Ändern einiger Zeilen sinnvoll sein.
Erweitern Sie die Verträglichkeitsmatrix (compatibility matrix). Zeigen Sie anhand eines Beispiels das Zusammenspiel von RIX mit anderen Sperrmodi und begründen Sie warum die obige Aussage („Sinnvoll beim Lesen einer ganzen Tabellen und ändern einiger Zeilen“) gilt. Geben Sie dazu an, wie ohne RIX-Modus gesperrt werden müsste und vergleichen Sie mit dem Sperrverhalten, wenn RIX zur Verfügung steht.

Lösung:

	IR	IX	R	X	RIX
IR	+	+	+	-	+
IX	+	+	-	-	-
R	+	-	+	-	-
X	-	-	-	-	-
RIX	+	-	-	-	-

RIX ist eine Kombination von R und IX, die in der im Beispiel beschriebenen Situation Vorteile hat.
Beispiel:
TA1: Lies die Datensätze aller Studierenden und erlasse allen die Semestergebühr, die eine Gesamtnote 1,0 im letzten Semester erzielt haben. (Annahme: es gibt Attribute „Semestergebühr“ und „Gesamtnote“.) Es werden also alle Sätze gelesen, wenige davon verändert. Unabhängig davon werden in der Studententabelle einige Datensätze gelesen, etwa um Name und Matrikelnr derjenigen auszugeben, die im 20. oder höheren Semester sind

(TA2) Der Zugriff erfolgt über einen Index, nicht durch Lesen und Überprüfen aller Datensätze wie bei TA1.
Ohne RIX benötigt TA1 exklusiven Lesezugriffzugriff (R-Lock auf Tabellenebene), ferner jeweils eine Schreibsperre, wenn sie auf zu ändernde Datensätze trifft. Also muss es auf Tabellenebene eine IX-Sperre geben. Dazu muss die R-Sperre auf Tabellenebene aufgegeben werden.

Im Beispiel schreibt TA1 eine Teilmenge der Tabelle und liest die gesamte Tabelle, TA2 liest eine Teilmenge. Beide Teilmengen sind möglicherweise disjunkt und damit problemlos nebenläufig ausführbar. Sollte es aber Datensätze geben, die in beiden Mengen enthalten sind (Gesamtnote 1,0 und 20. Semester oder höher) wird der Konflikt auf der unteren (Satz-) ebene erkannt und der Leser muss auf den Schreiber warten oder umgekehrt.

Im Anhang findet sich die Beschreibung der Sperren in der Literatur zu Oracle. Achtung: die Bezeichnungen unterscheiden sich!

Aufgabe 4 (Deadlockwahrscheinlichkeit) *5 Punkte*

a) In der Vorlesung wurde die Konfliktwahrscheinlichkeit für Transaktionen näherungsweise berechnet. Legen Sie das gleiche Modell zu Grunde und berechnen Sie die Verklemmungswahrscheinlichkeit für Transaktionen (Deadlock-Wahrscheinlichkeit). Machen Sie die vereinfachende Annahme, dass nur direkte Verklemmungen (Zyklen der Länge 2 im Wartegraphen) auftreten.

Lösung:

(Die Konfliktwahrscheinlichkeit wurde in der Vorlesung hergeleitet. Sie ist unten noch einmal angegeben.)

Eine Transaktion *T1* ist beteiligt an einer Verklemmung der Länge 2, wenn *T1* auf *irgendeine* andere Transaktion *T′* warten muss:

$W(T1 \text{ wartet}) = \frac{n \cdot k^2}{2 \cdot r}$, wobei n = Anzahl der Transaktionen, k = Anzahl der Sperren pro Transaktion und r = Anzahl der Datensätze in der Datenbank. Des weiteren gilt die Annahme, dass $n \cdot k < r$.

Für jede der n–1 anderen Transaktionen gilt dieselbe Wartewahrscheinlichkeit. Damit es zu einem Deadlock zwischen zwei Transaktionen kommt, müssen diese aber *aufeinander* warten: d.h. Wenn *T1* auf irgendeine Transaktion *T′* wartet, dann ist die Wahrscheinlichkeit, dass diese *T′* auf *T1* wartet, gegeben durch:

$$W(T1 \text{ wartet auf } T') = \frac{n \cdot k^2}{2 \cdot r} \cdot \frac{1}{n}$$

Dies gilt für ein festes *T1* und ein beliebiges *T′*. Die Wahrscheinlichkeit, dass T1 in einem Deadlock mit irgend einer anderen Transaktion *T′* ist, ist also

$$
\begin{aligned}
W(T1 \text{ ist im Deadlock}) &= W(T1 \text{ wartet auf } T') \cdot W(T' \text{ wartet auf } T1) \\
&= \frac{n \cdot k^2}{2 \cdot r} \cdot \frac{n \cdot k^2}{2 \cdot r} \cdot \frac{1}{n} \\
&= \frac{n \cdot k^4}{4 \cdot r^2}
\end{aligned}
$$

Für jede andere der n Transaktionen ist die Wahrscheinlichkeit genauso hoch, daher ist die Gesamtwahrscheinlichkeit dafür, dass zwei gegenseitig aufeinander warten (direkter Deadlock von zwei TA)

$$W(\text{Deadlock}) = \frac{n^2 \cdot k^4}{4 \cdot r^2 \cdot n}.$$

(korrigiert!) W(Deadlock) durch n dividiert!

Nachtrag: Herleitung der Wahrscheinlichkeit für einen Konflikt:
n = Anzahl der Transaktionen,
k = Anzahl der Sperren pro Transaktion und
r = Anzahl der Datensätze in der Datenbank.

Jede Transaktion fordert im Verlauf ihrer Ausführung insgesamt k Sperren an, deshalb wird angenommen, dass sie zu einem festgelegten Zeitpunkt $\frac{k}{2}$ Sperren besitzt.
Das gesamte System besteht aus n Transaktionen, daher gilt für das System, dass es zu einem bestimmten Zeitpunkt $\frac{nk}{2}$ Sperren hält. Wird eine neue Sperre angefordert, so betrifft diese Sperre einen der r Datensätze. Damit ist die Wahrscheinlichkeit, dass es zu einem Konflikt mit einem bestehenden Datensatz kommt, $\frac{nk}{2r}$. Da die zweite Transaktionen, die am Konflikt beteiligt ist, auch k Sperren anfordert, ist die Wahrscheinlichkeit für einen Konflikt $\frac{knk}{2r}$ oder $\frac{nk^2}{2r}$.

Anhang

RS: row share
RX: row exclusive
S: share
SRX: share row exclusive
X: exclusive
*Yes, if no conflicting row locks are held by another transaction. Otherwise, waits occur.

Table 13–3 Summary of Table Locks

SQL Statement	Mode of Table Lock	Lock Modes Permitted?				
		RS	RX	S	SRX	X
SELECT...FROM *table*...	none	Y	Y	Y	Y	Y
INSERT INTO *table* ...	RX	Y	Y	N	N	N
UPDATE *table* ...	RX	Y*	Y*	N	N	N
DELETE FROM *table* ...	RX	Y*	Y*	N	N	N
SELECT ... FROM *table* FOR UPDATE OF ...	RS	Y*	Y*	Y*	Y*	N
LOCK TABLE *table* IN ROW SHARE MODE	RS	Y	Y	Y	Y	N
LOCK TABLE *table* IN ROW EXCLUSIVE MODE	RX	Y	Y	N	N	N
LOCK TABLE *table* IN SHARE MODE	S	Y	N	Y	N	N
LOCK TABLE *table* IN SHARE ROW EXCLUSIVE MODE	SRX	Y	N	N	N	N
LOCK TABLE *table* IN EXCLUSIVE MODE	X	N	N	N	N	N

The following sections explain each mode of table lock, from least restrictive to most restrictive. They also describe the actions that cause the transaction to acquire a table lock in that mode and which actions are permitted and prohibited in other transactions by a lock in that mode.

Row Share Table Locks (RS) A row share table lock (also sometimes called a **subshare table lock, SS**) indicates that the transaction holding the lock on the table has locked rows in the table and intends to update them. A row share table lock is automatically acquired for a *table* when one of the following SQL statements is run:

```
SELECT ... FROM table ... FOR UPDATE OF ... ;
LOCK TABLE table IN ROW SHARE MODE;
```

A row share table lock is the least restrictive mode of table lock, offering the highest degree of concurrency for a table.

Permitted Operations: A row share table lock held by a transaction allows other transactions to query, insert, update, delete, or lock rows concurrently in the same *table. Therefore, other transactions can obtain simultaneous row share, row exclusive,* share, and share row exclusive table locks for the same table.

Prohibited Operations: A row share table lock held by a transaction prevents other transactions from exclusive write access to the same table using only the following statement:

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

Row Exclusive Table Locks (RX) A row exclusive table lock (also called a **subexclusive table lock, SX**) generally indicates that the transaction holding the lock has made one or more updates to rows in the table. A row exclusive table lock is acquired automatically for a *table* modified by the following types of statements:

```
INSERT INTO table ... ;
UPDATE table ... ;
DELETE FROM table ... ;
LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

A row exclusive table lock is slightly more restrictive than a row share table lock.

Permitted Operations: A row exclusive table lock held by a transaction allows other transactions to query, insert, update, delete, or lock rows concurrently in the same table. Therefore, row exclusive table locks allow multiple transactions to obtain simultaneous row exclusive and row share table locks for the same table.

Prohibited Operations: A row exclusive table lock held by a transaction prevents other transactions from manually locking the table for exclusive reading or writing. Therefore, other transactions cannot concurrently lock the table using the following

statements:

```
LOCK TABLE table IN SHARE MODE;
LOCK TABLE table IN SHARE EXCLUSIVE MODE;
LOCK TABLE table IN EXCLUSIVE MODE;
```

Share Table Locks (S) A share table lock is acquired automatically for the *table* specified in the following statement:

```
LOCK TABLE table IN SHARE MODE;
```

Permitted Operations: A share table lock held by a transaction allows other transactions only to query the table, to lock specific rows with `SELECT ... FOR UPDATE`, or to run `LOCK TABLE ... IN SHARE MODE` statements successfully. No updates are allowed by other transactions. Multiple transactions can hold share table locks for the same table concurrently. In this case, no transaction can update the table (even if a transaction holds row locks as the result of a `SELECT` statement with the `FOR UPDATE` clause). Therefore, a transaction that has a share table lock can update the table only if no other transactions also have a share table lock on the same table.

Prohibited Operations: A share table lock held by a transaction prevents other transactions from modifying the same table and from executing the following statements:

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
LOCK TABLE table IN EXCLUSIVE MODE;
LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

Share Row Exclusive Table Locks (SRX) A share row exclusive table lock (also sometimes called a **share-subexclusive table lock, SSX**) is more restrictive than a share table lock. A share row exclusive table lock is acquired for a *table* as follows:

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
```

Permitted Operations: Only one transaction at a time can acquire a share row exclusive table lock on a given table. A share row exclusive table lock held by a transaction allows other transactions to query or lock specific rows using `SELECT` with the `FOR UPDATE` clause, but not to update the table.

Prohibited Operations: A share row exclusive table lock held by a transaction prevents other transactions from obtaining row exclusive table locks and modifying the same table. A share row exclusive table lock also prohibits other transactions from obtaining share, share row exclusive, and exclusive table locks, which prevents other transactions from executing the following statements:

```
LOCK TABLE table IN SHARE MODE;
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
LOCK TABLE table IN ROW EXCLUSIVE MODE;
LOCK TABLE table IN EXCLUSIVE MODE;
```

Exclusive Table Locks (X) An exclusive table lock is the most restrictive mode of table lock, allowing the transaction that holds the lock exclusive write access to the table. An exclusive table lock is acquired for a *table* as follows:

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

Permitted Operations: Only one transaction can obtain an exclusive table lock for a table. An exclusive table lock permits other transactions only to query the table.

Prohibited Operations: An exclusive table lock held by a transaction prohibits other transactions from performing any type of DML statement or placing any type of lock on the table.