Übungsblatt 2

Adrian Schollmeyer

Aufgabe 1

Rücksetzbarkeit RC s heißt rücksetzbar, falls folgende Bedingung erfüllt ist:

$$(T_i \text{ liest von } T_j \text{ in } s) \wedge (c_i \in s) \implies (c_j \to_s c_i)$$

d. h. falls in s T_i auf Daten zugreift, die T_j modifiziert werden, muss vor einem Commit von c_i immer ein Commit von c_j kommen. Sollte T_i abgebrochen werden, d. h. $a_i \in s$, muss dies nicht gelten.

Ist ein Schedule nicht rücksetzbar, kann bspw. das Problem auftreten, dass theoretisch Veränderungen von Werten $x_1, \ldots, x_n$ auf Basis von Werten $y_1, \ldots, y_m$ committet werden können, wobei $y_1, \ldots, y_m$ noch gar nicht committet sind. Wenn jetzt die Transaktion, die $y_1, \ldots, y_m$ verändert hat (und auf deren Basis $x_1, \ldots, x_n$ verändert wurden), abgebrochen wird, erreicht man einen inkonsisteten Datenbankzustand, in dem ein Ergebnis einer Berechnung mit Eingabedaten vorliegt, die so nie in der Datenbank im persistenten Speicher existiert haben.

Rücksetzbarkeit alleine verhindert zwar, dass man einen Datenbankzustand auf Basis noch nicht committeter Daten erreicht, jedoch kann der Abbruch einer Transaktion, von der eine andere abhängt, dazu führen, dass auch die zweite Transaktion abgebrochen werden muss, da diese jetzt einen dirty read hat.

Vermeidung kaskadierender Abbrüche ACA Schedule s vermeidet kaskadierende Abbrüche, falls folgende Bedingung erfüllt ist:

$$(T_i \text{ liest } x \text{ von } T_j \text{ in } s) \implies (c_j \to_s r_i(x))$$

d. h. falls x in der Transaktion T_j modifiziert und in T_i gelesen wird, muss T_j committet werden, noch bevor x in T_i gelesen wird.

So kann ein Abbruch von T_j nicht dazu führen, dass T_i einen dirty read ausführt, da T_i jetzt in jedem Fall den Zustand liest, der auch der tatsächlich committete Datenbankzustand ist.

Striktheit ST Ein Schedule s heißt strict, falls folgende Bedingung gilt:

$$(w_j(x) \to_s p_i(x) \wedge j \neq i) \implies (a_j \to_s p_i(x) \vee c_j \to_s p_i(x), p \in \{r, w\})$$

d. h. eine Transaktion T_i darf also niemals auf Daten zugreifen, die T_j geschrieben hat, solange T_j noch nicht beendet (d. h. committet oder aborted) wurde.

Korrektheit Ein Schedule s ist korrekt, wenn der Effekt des Schedules s (Ergebnis der Ausführung des Schedules) äquivalent zu dem Effekt eines (beliebigen) seriellen Schedules s' bzgl. derselben Menge von Transaktionen ist (in Zeiten: $s \approx s'$).

Bei einem korrekten Schedule führt also die Tatsache, dass möglicherweise Operationen mehrerer Transaktionen in ihrer Reihenfolge miteinander verzahnt sind, nicht zu einer Veränderung des Ergebnisses (im Vergleich zu einem Schedule, bei dem die Transaktionen strikt hintereinander ausgeführt werden).

Versionierungsfunktion Die Versionierungsfunktion h bildet Leseoperationen auf Objekten in Leseooperationen auf Versionen ab:

$$\begin{aligned} h(w_i(x)) &= w_i(x_i) \\ h(r_i(x)) &= r_i(x_j) \quad \exists \text{ geeignetes } j \\ h(a_i) &= a_i \\ h(c_i) &= c_i \end{aligned}$$

Zu allen Operationen werden also die passenden Versionen rausgesucht, auf denen diese Operationen basiseren. Für Schreiboperationen, Commits und Aborts ist das dabei einfach die Transaktion T_i selbst, in denen diese stattfinden. Für Leseoperationen wird die Transaktion T_j gesucht, die das Objekt zuletzt geschrieben hat.

Damit immer eine solche Transaktion T_j gefunden werden kann, wird zu Beginn immer eine Initialisierungstransaktion T_0 angenommen, die alle beteiligten Objekte schreibt.

Mehrversionen-Schedule Ein Mehrversionen-Schedule repräsentiert die Ausführung durch den Transaktionsmanager auf einer Datenbank mit Mehrversionen. Für einen *vollständigen* Mehrversionen-Schedule m für eine Menge von Transaktionen $\mathbf{T} = \{T_1, \ldots, T_m\}$ gilt:

(1) $op(m) = h(\bigcup_{i=1}^{n} op(T_i))$.

Alle Operationen in M sind also versioniert.

(2) Für alle $T_i \in \mathbf{T}$ und alle Operationen $p, q \in op(\mathbf{T})$ gilt, dass die partielle Ordnung der Operationen auch für die versionierten Operationen erhalten bleibt: $p \to_m q \implies h(p) \to_m h(q)$

(3) Transaktionen können nur Versionen lesen, die bereits erzeugt wurden: Wenn $h(r_j(x)) = r_j(x_i)$, dann $w_i(x_i) \to_m r_j(x_i)$.

(4) Zur Erhaltung der reflexiven Liest-von-Relation muss die Transaktion, die x schreibt, bevor sie liest, die zuvor erzeugte Version von x lesen: Wenn $w_i(x) \to_i r_i(x)$, dann $h(r_i(x)) = r_i(x_i)$.

(5) Wenn $h(r_j(x)) = r_j(x_i)$ mit $i \neq j$ und c_j ist in m, dann ist c_i in m und es gilt $c_i \to_m c_j$ (Rücksetzbarkeit).

Greift T_j also auf Daten von T_i zu, muss T_i vor T_j committet werden (sofern T_j committet wird).

Einversionen-Sicht Die Einversionen-Sicht s einen MV-Schedules m ist gültig, wenn sie über der gleichen Menge von Transaktionen wie m definiert ist und eine bijektive Versionierungsfunktion $h : op(m) \to op(s)$ existiert.

m und s sind äquivalent, wenn ihre Liest-von-Relation gleich sind.

Mehrversionen-Sichtserialisierbarkeit MVSR Ein MV-Schedule m ist mehrversionen-sichtserialisierbar (in MVSR) genau dann, wenn eine Versionsordnung $\ll$ existiert, sodass der MV-Serialsierbarkeitsgraph $MVSG(m, \ll)$ azyklisch ist.

Ein MV-Schedule m ist mehrversionen-sichtserialisierbar genau dann, wenn $CP(m)$ äquivalent zu einem 1-seriellen MV-Schedule ist. Ein serieller MV-Schedule ist 1-seriell, wenn für alle i, j und x gilt: Wenn T_i das Objekt x von T_j liest, dann ist entweder $i = j$ oder T_j ist die letzte Transaktion vor T_i, die irgendeine der Versionen von x schreibt.

Es ist also wieder gefordert, dass m sich so verhält als würde man alle Transaktionen nacheinander ausführen.

Aufgabe 2

(a)

Präfix-Commit-Abgeschlossenheit fordert, dass Eigenschaften E, die für s gelten, auch für alle Präfixe von s gelten und diese auch für $CP(s)$ gelten (d. h. auch dann, wenn einige Transaktionen abgebrochen werden).

Ein sichtserialisierbarer Schedule wäre bspw.

$$s = r_1(x)\, w_1(x)\, r_2(x)\, r_3(x)\, w_2(x)\, r_4(x)\, a_2 c_1 c_3 c_4 \tag{1}$$

mit

$$CP(s) = r_1(x)\, w_1(x)\, r_3(x)\, r_4(x)\, c_1 c_3 c_4 \tag{2}$$

Dabei liest T_3 in s von T_2, aber T_2 wird später abgebrochen. Damit ist $RF(s) \neq RF(CP(s))$, womit die Commit-Abgeschlossenheit verletzt ist. Somit ist VSR nicht präfix-commit-abgeschlossen.

(b)

CSR ist präfix-commit-abgeschlossen, d. h. alle Eigenschaften E, die für einen konfliktserialisierbaren Schedule s gelten, gelten auch für alle Präfixe von s und auch für $CP(s)$.

Bei der Betrachtung der Konfliktäquivalenz werden lediglich committete Transaktionen berücksichtigt. Damit gelten Eigenschaften, die für s gelten, auch für $CP(s)$.

Aufgabe 3

(a)

Konflikte:

- $w_4(a) \to_s w_2(a)$, also $(T_4, T_2) \in C(s)$
- $w_3(c) \to_s r_1(c)$, also $(T_3, T_1) \in C(s)$
- $w_4(b) \to_s r_3(b)$, also $(T_4, T_3) \in C(s)$

Die restlichen Konflikte vergrößern $C(s)$ nicht. Der daraus resultierende Konfliktgraph ist azyklisch, also ist $s \in CSR$.

(b)

(c)

Ein Abbruch von T_4 hat einen Dirty Read $r_3(b)$ nach vorigem $w_4(b)$ zur Folge. Entsprechend ist T_3 auch abzubrechen. Der Konflikt $w_4(a) \to_s w_2(s)$ kann zur Folge haben, dass a wieder auf den Wert vor $w_4(a)$ zurückgesetzt wird, obwohl zuletzt $w_2(a)$ ausgeführt und committet wurde.

Aufgabe 4

Es gilt seriell $\subset ST \subset ACA \subset RC$. Somit $s_i \in$ seriell $\implies s_i \in ST$, $s_i \in ST \implies s_i \in ACA$ und $s_i \in ACA \implies s_i \in RC$.

- $s_1 \notin ACA$, da $r_2(x)$ vor c_1 kommt, aber $s_1 \in RC$, da $c_1 \to_s c_s$.
- $s_2 \in ST$, da $w_1(x) \to_s w_2(x)$, aber auch $c_1 \to_s w_2(x)$. Jedoch s_2 nicht seriell.
- $s_3 \notin RC$, da $w_1(y) \to_s r_2(y)$, aber $c_2 \to_s c_1$.
- $s_4 \notin ST$, da $w_2(x) \to_s w_1(x)$, aber nicht $c_2 \to_s w_1(x)$. Jedoch $s_4 \in ACA$, da keine Leseoperationen stattfinden.
- $s_5 \notin RC$, da $w_2(y) \to_s r_1(y)$, aber $c_1 \to_s c_2$.
- s_6 ist seriell.
- $s_7 \notin ST$, da $w_2(x) \to_s w_1(x)$, aber nicht $c_2 \to_s w_1(x)$. Jedoch $s_7 \in ACA$, da $w_1(y) \to_s r_2(y)$ und $c_1 \to_s r_2(y)$.

Aufgabe 5

Einversionen-Schedules m_i' zu m_i, $i \in \{1,2,3\}$:

$$m_1' = w_0(x)\, w_0(y)\, w_0(z)\, c_0 r_3(x)\, w_3(x)\, c_3 w_1(x)\, c_1 r_2(x)\, w_2(y)\, w_2(z)\, c_2 \tag{3}$$
$$m_2' = w_0(x)\, w_0(y)\, c_0 w_1(x)\, c_1 r_3(x)\, w_3(x)\, r_2(x)\, c_3 w_2(y)\, c_2 \tag{4}$$
$$m_3' = w_0(x)\, w_0(y)\, c_0 w_1(x)\, c_1 r_2(x)\, w_2(y)\, c_2 r_3(y)\, w_3(x)\, c_3 \tag{5}$$

Liest-von-Relationen:

$$RF_M(m_1) = \{(T_0, x, T_3), (T_1, x, T_2)\} \tag{6}$$
$$RF_M(m_1') = \{(T_0, x, T_3), (T_1, x, T_2)\} \tag{7}$$
$$RF_M(m_2) = \{(T_1, x, T_3), (T_1, x, T_2)\} \tag{8}$$
$$RF_M(m_2') = \{(T_1, x, T_3), (T_3, x, T_2)\} \tag{9}$$
$$RF_M(m_3) = \{(T_1, x, T_2), (T_0, y, T_3)\} \tag{10}$$
$$RF_M(m_3') = \{(T_1, x, T_2), (T_2, y, T_3)\} \tag{11}$$

$m_1 \in MVSR$, da MVSG azyklisch.