Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Dr. G. Alonso *C. Hagen, B. Kemme*

AR 1: Transaktionsverwaltung

Theoretische Übung 2 — Beispiellösung

Aufgabe 1: Blind Writes

Ein Blind Write ist ein Write einer Transaktion auf ein Objekt, das diese Transaktion vorher nicht gelesen hat.

Beweisen Sie unter der Annahme, dass Transaktionen keine Blind Writes enthalten (d.h. wenn $w_i(x) \in T_i$ dann gilt $r_i(x) \in T_i$ und $r_i(x) <_i w_i(x)$), folgende Aussage:

Ein Schedule ist genau dann konfliktserialisierbar, wenn er viewserialisierbar ist.

Lösung:

Z.z.: $S \in CPSR \Rightarrow S \in VSR$: *klar nach Satz 5 der Vorlesung.*

Z.z.: $S \in VSR \Rightarrow S \in CPSR$,
d.h. z.z.: $\exists S_{ser} | \forall o_i(x), o_j(x) \in A(S)$ $(=A(S_{ser}))$, $x \in DB$ *gilt:*

$(o_i(x), o_j(x)) \in CON(S)$ *(d.h.* $(o_i(x), o_j(x))$ *Konfliktpaar in S)*
$\Leftrightarrow (o_i(x), o_j(x)) \in CON(S_{ser})$

$S \in VSR$, *d.h.* $\exists S_{ser}$ *mit* $RF(\overline{S}) = RF(\overline{S_{ser}})$

„$\subseteq$“:

1. *Sei* $(w_i(x), r_j(x)) \in CON(S) \Rightarrow$
   a) $(T_i, x, T_j) \in RF(\overline{(}S)) = RF(\overline{S_{ser}})$
      $\Rightarrow (w_i(x), r_j(x)) \in CON(S_{ser})$: $\surd$
   b) $\exists\ w_{k_1}(x), \ldots, w_{k_n}(x)$ *mit* $w_i(x) <_S w_{k_1}(x) <_S \ldots <_S w_{k_n}(x) <_S r_j(x)$
      $\Rightarrow$ *(wegen keiner Blind Writes)* $\exists r_{k_1}(x), \ldots, r_{k_n}(x)$ *mit* $w_i(x) <_S r_{k_1}(x) <_S w_{k_1}(x) <_S \ldots <_S r_{k_n}(x) <_S w_{k_n}(x) <_S r_j(x)$
      $\Rightarrow (T_i, x, T_{k_1}), (T_{k_1}, x, T_{k_2}), \ldots, (T_{k_n}, x, T_j) \in RF(\overline{S}) = RF(\overline{S_{ser}})$
      $\Rightarrow w_i(x) <_{S_{ser}} r_{k_1}(x) <_{S_{ser}} w_{k_1}(x) <_{S_{ser}} \ldots <_{S_{ser}} r_{k_n}(x) <_{S_{ser}} w_{k_n}(x) <_{S_{ser}} r_j(x)$
      $\Rightarrow (w_i(x), r_j(x)) \in CON(S_{ser})$: $\surd$
2. *Voraussetzung:* $(r_i(x), w_j(x)) \in CON(S)$.
   Annahme: $(r_i(x), w_j(x)) \notin CON(S_{ser})$
   $\Rightarrow (w_j(x), r_i(x)) \in CON(S_{ser})$
   $\Rightarrow \exists k_1, \ldots, k_n$ *mit:* $(T_j, x, T_{k_1}), (T_{k_1}, x, T_{k_2}), \ldots, (T_{k_n}, x, T_i) \in RF(\overline{S_{ser}}) = RF(\overline{S})$
   $\Rightarrow w_j(x) <_S r_i(x)$
   $\Rightarrow (w_j(x), r_i(x)) \in CON(S)$. *Widerspruch zur Voraussetzung!*

3. *Sei* $(w_i(x), w_j(x)) \in CON(S)$
   $\Rightarrow \exists r_j(x)$ *und* $k_1, \ldots, k_n$*:* $w_i(x) <_S r_{k_1}(x) <_S w_{k_1}(x) <_S \ldots <_S r_{k_n}(x) <_S w_{k_n}(x) <_S r_j(x) <_S w_j(x)$
   $\Rightarrow$ *(wieder über RFs analog zu 1)* $w_i(x) <_{S_{ser}} r_{k_1}(x) <_{S_{ser}} w_{k_1}(x) <_{S_{ser}} \ldots <_{S_{ser}} r_{k_n}(x) <_{S_{ser}} w_{k_n}(x) <_{S_{ser}} r_j(x) <_{S_{ser}} w_j(x)$
   $\Rightarrow (w_i(x), w_j(x)) \in CON(S_{ser})$

„$\supseteq$“*:*

analog.

Aufgabe 2: Inkrementelle Erkennung der Serialisierbarkeit

$$\begin{aligned} T_1 &= < r_1^1(a)\ r_2^1(c)\ r_3^1(b)\ w_4^1(b)\ w_5^1(c) > \\ T_2 &= < r_1^2(c)\ r_2^2(a)\ w_3^2(a)\ r_4^2(b) > \\ S &= < r_1^2(c)\ r_1^1(a)\ r_2^1(c)\ r_2^2(a) \ldots > \end{aligned}$$

Versuchen Sie den Schedule der Transaktionen T_1, T_2 so fortzusetzen, dass er

a) viewserialisierbar ist

b) konfliktserialisierbar ist

Entscheiden Sie in beiden Fällen, ob dies möglich ist und geben Sie das Entscheidungsvorgehen an.

Lösung:

a) **Viewserialisierbarkeit**

Viewserialisierbarkeit kann grundsätzlich nicht inkrementell entschieden werden, da die Klasse VSR **nicht monoton** *ist. Eine spätere Aktion kann einen Schedule der aktuell nicht viewserialisierbar erscheint wieder “korrigieren”.*

Die Lösung findet sich hier *leicht analytisch. Der Schedule beginnt mit*

$$S = < r_1^2(c)\ r_1^1(a)\ r_2^1(c)\ r_2^2(a) \ldots >$$

Beide Transaktionen haben also a und c von T_0 *gelesen. Danach schreibt* T_2 *das Objekt a und* T_1 *das Objekt c. In der seriellen Folge* $T_1 < T_2$ *liest* T_2 *aber c von* T_1*, in der Reihenfolge* $T_2 < T_1$ *liest* T_1 *a von* T_2*. Damit kann der Schedule nicht viewserialisierbar sein.*

b) **Konfliktserialisierbarkeit**

Im Prinzip kann aus a) gefolgert werden, dass S auch nicht konfliktserialisierbar fortgesetzt werden kann. Dies lässt sich auch leicht analytisch zeigen: Da beide Transaktionen ein Objekt (a resp. c) schreiben wollen, welches die andere schon gelesen hat, entsteht zwansgläufig ein Zyklus im Abhängigkeitsgraphen.

Aufgabe 3: Rücksetzbarkeit / Striktheit

Zeigen Sie, dass die strikten Schedules eine echte Teilmenge der kaskadenfreien Schedules sind. Finden Sie dazu ein Beispiel für einen Schedule, der kaskadenfrei, aber nicht strikt ist.

Striktheit ist unabhängig von Serialisierbarkeit: Konstruieren Sie einen Schedule der strikt ist, aber nicht konfliktserialisierbar.

Lösung:

Ein Schedule ist kaskadenfrei, wenn eine TA nur "comittete" Werte liest. Er ist strikt, wenn er kaskadenfrei ist und nur "comittete" Werte überschreibt. Ein Beispiel ist also über einen "reinen Schreiber" leicht zu konstruieren.

$$\begin{aligned} S_1 &= < r_1^1(a)\ w_2^1(a)\ w_1^2(a)\ c^1\ c^2 > \\ S_1 &\in ACA \\ S_1 &\notin ST \end{aligned}$$

Die Aktion $w_1^2(a)$ verletzt die Bedingung der Striktheit.

$$\begin{aligned} S_2 &= < r_1^1(a)\ r_1^2(a)\ w_2^2(a)\ c^2\ r_2^1(a)\ c^1 > \\ S_2 &\in ST \\ S_2 &\notin CPSR \end{aligned}$$

$r_1^1(a)$, $r_1^2(a)$ und $w_2^2(a)$ arbeiten auf den ("comitteten") Ausgangswerten in der Datenbank. $r_2^1(a)$ liest den "comitteten" Wert von T_2. Daher ist S_2 strikt.

Wegen $r_1^1(a) \ldots w_2^2(a) \ldots r_2^1(a)$ ist S_2 sicher nicht konfliktserialisierbar.

Aufgabe 4: Sperrverfahren

a) Jeder 2PL – Schedule ist konfliktserialisierbar. Zeigen Sie, dass die Umkehrung, d.h. $S \in \text{CPSR} \Rightarrow S \in \text{2PL}$, **nicht** gilt.

Lösung:

Zu zeigen ist, dass es einen Schedule gibt, der CPSR ist, aber nicht durch das 2PL-Protokoll entstehen kann. Der folgende Schedule ist ein Beispiel:

$S = < w_1(x)\ r_2(x)\ c_2\ \ w_1(y)\ c_1 >$

b) Betrachten Sie ein Sperrprotokoll, in dem keine Lesesperren gesetzt werden und in dem keine Lock-Konversion durchgeführt wird (wenn ein Datenobjekt geschrieben werden soll, wird eine Schreibsperre erworben). Schreibsperren werden bis zum Commit der Transaktion gehalten.

Sind mit diesem Protokoll *lost updates* möglich? Können *Dirty Reads* auftreten? Sind Leseoperationen wiederholbar (*repeatable read*) ?

Lösung:

Es kann lost updates geben, da Transaktionen keine Lesesperren halten. Der folgende Schedule ist ein Beispiel:

$$S = r_1[x] r_2[x] w_1[x] c_1 w_2[x] c_2$$

Es kann keine dirty reads geben, da Updates erst beim Commit einer Transaktion Es kann jedoch nichtwiederholbare Leseoperationen geben, da keine Lesesperren erworben werden und damit zwischen zwei Leseoperationen der gleichen Transaktion Änderungen einer anderen Transaktion vorkommen können.

c) Betrachten Sie ein modifiziertes 2PL-Protokoll, in dem Lesesperren zu Beginn der Schrumpfungsphase freigegeben werden, sobald die Leseoperationen beendet sind. Schreibsperren werden bis zum Commit der Transaktion gehalten.

Garantiert dieses Protokoll Serialisierbarkeit? Können *Dirty Reads* auftreten? Sind Leseoperationen wiederholbar (*repeatable read*) ? Sind alle so entstehenden Schedules strikt?

Lösung:

Ja, dies garantiert serialisierbare Schedules. Das Protokoll enspricht den 2PL-Regeln. Da die Schedules serialisierbar sind, können weder inrepeatable reads, noch lost updates oder dirty reads auftreten. Sobald die Objekte gelesen wurden und keine weiteren Sperren mehr benötigt werden, können die Lesesperren freigegeben werden. Transaktionen, die die gelesenen Daten modifizieren, werden nach der lesenden Transaktion serialisiert. Das Protokoll garantiert strikte Schedules, da keine uncommitteten Daten gelesen oder überschrieben werden.

d) Betrachten Sie ein modifiziertes 2PL-Protokoll, bei dem in der Schrumpfungsphase Lock-Konversionen zugelassen sind, d.h. eine S-Sperre kann in eine X-Sperre umgewandelt werden. Zeigen Sie durch ein Gegenbeispiel, dass ein derart modifiziertes 2PL-Protokoll Konfliktserialisierbarkeit nicht mehr garantiert.

Lösung:

Das 2PL-Protokoll sichert Konfliktserialisierbarkeit zu. Um die Verletzung der Konfliktserialisierbarkeit im modifizierten 2PL-Protokoll zu zeigen, muss nun die Lockkonversion einen Zyklus im Abhängigkeitsgraphen erzeugen.

Die Transaktionen müssen mindestens auf zwei Objekten x und y operieren, da bei zyklischer Abhängigkeit auf einem Objekt keine Zweiphasigkeit möglich ist. Ferner muss eine Transaktion ein Objekt lesen und schreiben (Lockkonversion). Nehmen wir an, T1 lese und schreibe das Objekt x und lese zwischendurch das Objekt y, so kann T2 das Objekt x lesen und muss das Objekt y schreiben, damit die nötigen Konfliktpaare für den Zyklus im Serialisierbarkeitsgraphen vorhanden sind. Wir können somit die folgenden Transaktionen

$T1 = < r_1^1(x)\ r_2^1(y)\ w_3^1(x) >$

$T2 = < r_1^2(x)\ w_2^2(y) >$

aufstellen und müssen noch einen Schedule bestimmen, der einerseits einen Zyklus erzeugt, aber andererseits noch zweiphasig ablaufen kann. Nach einigen Überlegungen erhält man den Schedule

$S = < r_1^1(x)\ r_1^2(x)\ r_2^1(y)\ w_2^2(y)\ w_3^1(x) >,$

der sowohl einen Zyklus im Serialisierbarkeitsgraphen enthält,

als auch einen Ablauf im modifizierten 2PL-Protokoll ermöglicht.