AG Datenbanken und Informationssysteme

Wintersemester 2006 / 2007

Prof. Dr.-Ing. Dr. h. c. Theo Härder
Fachbereich Informatik
Technische Universität Kaiserslautern

http://wwwdvs.informatik.uni-kl.de

9. Übungsblatt

Für die Übung am Donnerstag, **11. Januar 2007**,
von 15:30 bis 17:00 Uhr in 13/222.

Aufgabe 1: Verklemmungen

Gegeben sind die Transaktionen T_1, T_2, T_3, T_4 und T_5 sowie die Datenelemente A, B, C, D, E und F. Es gelte das Zwei-Phasen-Sperr-Protokoll. Dabei bedeutet $T_i(R, A)$, dass die Transaktion T_i eine R-Sperre auf dem Datenelement A hält bzw. anfordert. $T_i(X, A)$ bedeute, dass die Transaktion T_i eine X-Sperre auf dem Datenelement A hält bzw. anfordert.

Folgende Sperren werden aktuell durch die Transaktionen gehalten:

$T_1(X, A)$, $T_1(R, B)$, $T_1(R, E)$, $T_2(R, B)$, $T_3(R, C)$, $T_3(R, E)$, $T_4(R, C)$, $T_4(X, D)$, $T_4(R, E)$, $T_5(X, F)$

Die einzelnen Transaktionen fordern jeweils folgende Sperren an, um mit der Bearbeitung fortfahren zu können:

$T_1(X, C)$, $T_2(R, F)$, $T_3(X, B)$, $T_4(R, A)$, $T_5(X, D)$

a) Zeichnen Sie den Wartegraphen dieser Transaktionen.

b) Begründen Sie anhand Ihres Wartegraphen, warum eine Verklemmung vorliegt.

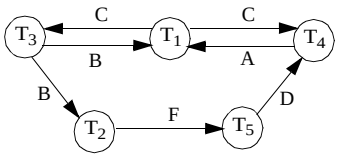
c) Lösen Sie diese Verklemmung durch Zurücksetzen einer Transaktion auf.

Lösung:

a) Folgende Sperren werden zu Beginn gehalten:

$T_1(X,A)$	$T_1(R,B)$	$T_3(R,C)$	$T_4(X,D)$	$T_1(R,E)$	$T_5(X,F)$
	$T_2(R,B)$	$T_4(R,C)$		$T_3(R,E)$	
				$T_4(R,E)$	

Damit ergibt sich der Wartegraph:

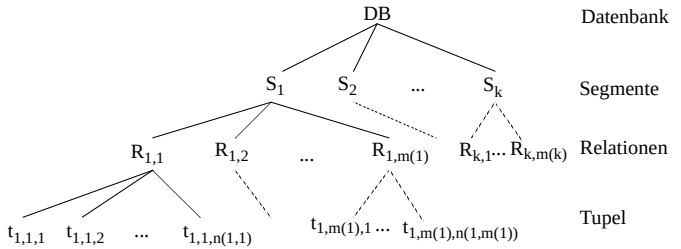


b) Im Wartegraph existieren Zyklen (T_3 -> T_1 -> T_3, T_4 -> T_1 -> T_4, usw.) und damit auch Verklemmungen.

c) Zurücksetzen von T_1.

Aufgabe 2: Sperrprotokolle in Datenbankystemen

Die Objekte der Datenbank seien in folgender Weise hierarchisch angeordnet:



Diese Darstellung zeigt, dass sich die Relation $R_{1,1}$ aus den Tupeln $t_{1,1,1}$, $t_{1,1,2}$, ..., $t_{1,1,n(1,1)}$, das Segment S_1 aus den Relationen $R_{1,1}$, $R_{1,2}$, ..., $R_{1,m(1)}$ und die DB aus den Segmenten S_1, S_2, ..., S_k zusammensetzen.

Die einzelnen Transaktionen T_i fordern in folgender zeitlicher Reihenfolge in Abständen von jeweils einer Zeiteinheit Objekte zum Lesen (R) oder Schreiben (X) an. Schreibsperren werden 10 und Lesesperren 20 Zeiteinheiten lang gehalten.

Z_1: T_1 <-- $t_{1,1,1}$ in X
Z_2: T_2 <-- $t_{1,1,2}$ in R
Z_3: T_3 <-- $R_{1,1}$ in R
Z_4: T_4 <-- $R_{1,2}$ in X
Z_5: T_5 <-- S_1 in R
Z_6: T_6 <-- S_2 in X
Z_7: T_7 <-- DB in R

Freigabe der Schreibsperren, wenn sie sofort gewährt werden, zu den Zeitpunkten

Z_{11}: T_1 --> X von $t_{1,1,1}$
Z_{14}: T_4 --> X von $R_{1,2}$
Z_{16}: T_6 --> X von S_2

a) Wie sehen die Sperrprotokolle und Wartebeziehungen bei Partitionssperren jeweils aus, wenn die Sperreinheit ein Tupel, eine Relation, ein Segment bzw. die Datenbank ist?

b) Wie sehen die Sperrprotokolle und Wartebeziehungen bei hierarchischem Sperrmodus mit einer allgemeinen Anwartschaftssperre (I) aus?

c) Wie viele Sperranforderungen sind nötig?

d) Wie oft muss dabei eine Transaktion deaktiviert und in eine Warteschlange gebracht werden?

e) Nach wie vielen Zeiteinheiten sind alle Transaktionen bearbeitet?

f) Wie hoch ist die durchschnittliche Wartezeit?

g) Es seien zwei Arten von speziellen Anwartschaftssperren IR und IX vorhanden. Wie ändern sich die Sperrprotokolle?

h) Berechnen Sie jeweils die Anzahl der Sperranforderungen und der Deaktivierungen, die insgesamt verbrauchten Zeiteinheiten sowie die durchschnittlichen Wartezeiteinheiten.

Lösung:

a) Sperrprotokolle und Wartebeziehungen:

Sperreinheit ist das Tupel.
T_1: $X(t_{1,1,1})$
T_2: $R(t_{1,1,2})$
T_3 fordert $R(t_{1,1,1})$ und muss warten wg. T_1
T_4: $X(t_{1,2,1})$, $X(t_{1,2,2})$,..., $X(t_{1,2,n(1,2)})$
T_5 fordert $R(t_{1,1,1})$ und muss warten wg. T_1
T_6: $X(t_{2,1,1})$, $X(t_{2,1,2})$,..., $X(t_{2,1,n(2,1)})$,$X(t_{2,2,1})$, $X(t_{2,2,2})$,..., $X(t_{2,2,n(2,2)})$, ..., $X(t_{2,m(2),1})$, $X(t_{2,m(2),2})$,..., $X(t_{2,m(2),n(2,m(2))})$
T_7 fordert $R(t_{1,1,1})$ und muss warten wg. T_1

Warteschlange vor $t_{1,1,1}$: $R(T_3)$, $R(T_5)$, $R(T_7)$

Nach Freigabe der X-Sperre von T_1 auf $t_{1,1,1}$:
T_3: $R(t_{1,1,1})$, $R(t_{1,1,2})$,..., $R(t_{1,1,n(1,1)})$
T_5: $R(t_{1,1,1})$, $R(t_{1,1,2})$,..., $R(t_{1,1,n(1,1)})$, fordert $R(t_{1,2,1})$ und muss warten wg. T_4
T_7: $R(t_{1,1,1})$, $R(t_{1,1,2})$,..., $R(t_{1,1,n(1,1)})$, fordert $R(t_{1,2,1})$ und muss warten wg. T_4

Warteschlange vor $t_{1,2,1}$: $R(T_5)$, $R(T_7)$

Nach Freigabe der X-Sperre von T_4 auf $R_{1,2}$:
T_5: $R(t_{1,2,1})$, $R(t_{1,2,2})$,..., $R(t_{1,2,n(1,2)})$, ..., $R(t_{1,m(1),1})$, $R(t_{1,m(1),2})$,..., $R(t_{1,m(1),n(1,m(1))}$
T_7: $R(t_{1,2,1})$, $R(t_{1,2,2})$,..., $R(t_{1,2,n(1,2)})$, ..., $R(t_{1,m(1),1})$, $R(t_{1,m(1),2})$,..., $R(t_{1,m(1),n(1,m(1))}$, fordert $R(t_{2,1,1})$ und muss warten wg. T_6

Warteschlange vor $t_{2,1,1}$: $R(T_7)$

Nach Freigabe der X-Sperre von T_6 auf S_2:
T_7: $R(t_{2,1,1})$, $R(t_{2,1,2})$,..., $R(t_{2,1,n(2,1)})$, $R(t_{2,2,1})$,..., $R(t_{k,m(k),n(k,m)})$

Sperreinheit ist die Relation.
T_1: $X(R_{1,1})$
T_2 fordert $R(R_{1,1})$ und muss warten wg. T_1
T_3 fordert $R(R_{1,1})$ und muss warten wg. T_1
T_4: $X(R_{1,2})$
T_5 fordert $R(R_{1,1})$ und muss warten wg. T_1
T_6: $X(R_{2,1})$, $X(R_{2,2})$,..., $X(R_{2,m(2)})$
T_7 fordert $R(R_{1,1})$ und muss warten wg. T_1

Warteschlange vor $R_{1,1}$: $R(T_2)$, $R(T_3)$, $R(T_5)$, $R(T_7)$

Nach Freigabe der X-Sperre von T_1 auf $R_{1,1}$:
T_2: $R(R_{1,1})$
T_3: $R(R_{1,1})$
T_5: $R(R_{1,1})$, fordert $R(_{1,2})$ und muss warten wg. T_4
T_7: $R(R_{1,1})$, fordert $R(_{1,2})$ und muss warten wg. T_4

Warteschlange vor $R_{1,2}$:$R(T_5)$, $R(T_7)$

Nach Freigabe der X-Sperre von T_4 auf $R_{1,2}$:
T_5: $R(R_{1,2})$,..., $R(R_{1,m(1)})$
T_7: $R(R_{1,2})$,..., $R(R_{1,m(1)})$, fordert $R(R_{2,1})$ und wartet wg. T_6

Warteschlange vor $R_{2,1}$:$R(T_7)$

Nach Freigabe der X-Sperre von T_6 auf S_2:
T_7: $R(R_{2,1})$, $R(R_{2,2})$,..., $R(R_{2,m(2)})$,..., $R(R_{k,m(k)})$

Sperreinheit ist das Segment.
T_1: $X(S_1)$
T_2 fordert $R(S_1)$ und wartet wg. T_1
T_3 fordert $R(S_1)$ und wartet wg. T_1
T_4 fordert $X(S_1)$ und wartet wg. T_1
T_5 fordert $R(S_1)$ und wartet wg. T_1
T_6: $X(S_2)$
T_7 fordert $R(S_1)$ und wartet wg. T_1

Warteschlange vor S_1:$R(T_2)$, $R(T_3)$, $X(T_4)$, $R(T_5)$, $R(T_7)$

Nach Freigabe der X-Sperre von T_1 auf S_1:
T_2: $R(S_1)$
T_3: $R(S_1)$
T_4 fordert $X(S_1)$ und wartet wg. T_2, T_3
T_5: $R(S_1)$
T_7: $R(S_1)$, fordert $R(S_2)$ und wartet wg. T_6

Nach Freigabe der X-Sperre von T_6 auf S_2:
T_7: $R(S_2)$, $R(S_3)$, ..., $R(S_k)$

Nach Freigabe der R-Sperren von T_2, T_3, T_5 und T_7:
T_4: $X(S_1)$

4. Sperreinheit ist die Datenbank.
T_1: X(DB)

Warteschlange für DB:$R(T_2)$, $R(T_3)$, $X(T_4)$, $R(T_5)$, $X(T_6)$, $R(T_7)$

Nach Z_{11} (Gewährung aller möglichen Sperren):
T_2: R(DB)
T_3: R(DB)
T_5: R(DB)
T_7: R(DB)

Warteschlange für DB:$X(T_4)$, $X(T_6)$
Danach Abarbeitung von zunächst T_4, dann T_6. Sollten davor neue Lesetransaktionen starten (was sehr wahrscheinlich ist), passiert es schnell, dass T_4 und T_6 „aushungern“.

b) Wie sehen die Sperrprotokolle und Wartebeziehungen bei hierarchischem Sperrmodus mit einer allgemeinen Anwartschaftssperre (I) aus?

c) Wie viele Sperranforderungen sind nötig?

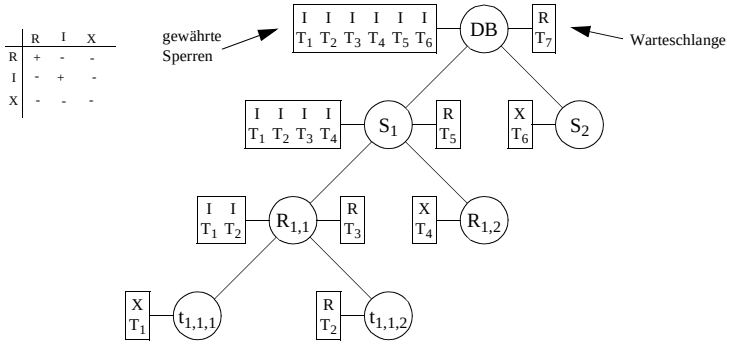
d) Wie oft muss dabei eine Transaktion deaktiviert und in eine Warteschlange gebracht werden?

e) Nach wie vielen Zeiteinheiten sind alle Transaktionen bearbeitet?

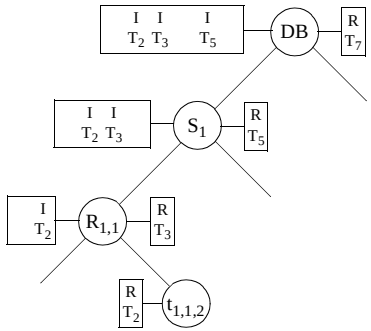
f) Wie hoch ist die durchschnittliche Wartezeit?

b) + c) + d) + e) + f): **Hierarchische Sperrprotokolle bei allgemeiner Anwartschaftssperre**

Folgende Sperren werden gewährt:



Zum Zeitpunkt Z_{16} keine weitere Gewährung von Sperren aus den Warteschlangen möglich. Es ergibt sich folgende Situation:



Anzahl der Sperranforderungen: Insgesamt 19 Sperren angefordert

Anzahl der Deaktivierungen: Insgesamt 3 Transaktionen in Wartestellung

Transaktion	Dauer	Zeiteinheiten	Wartezeiten	Bemerkung
T_1	Z_1 - Z_{11}	10	0	
T_2	Z_2 - Z_{22}	20	0	
T_3	Z_3 - Z_{42}	39	19	„läuft“ erst nach Ende T_2
T_4	Z_4 - Z_{14}	10	0	
T_5	Z_5 - Z_{62}	57	37	„läuft“erst nach Ende T_3
T_6	Z_6 - Z_{16}	10	0	
T_7	Z_7 - Z_{82}	75	55	„läuft“ erst nach Ende T_5

Gesamtdauer: 82 Zeiteinheiten

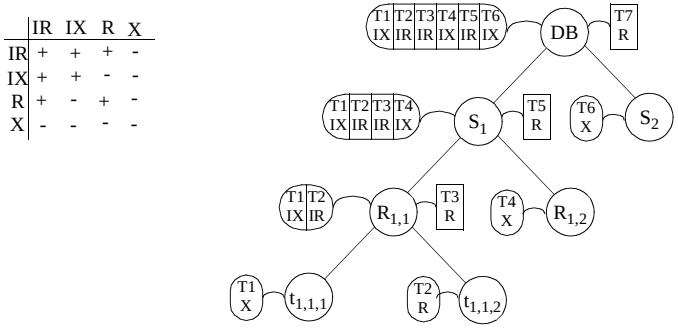
Durchschnittliche Wartezeit:

$$W = \frac{19 + 37 + 55}{7} = \frac{111}{7} = 15{,}9$$

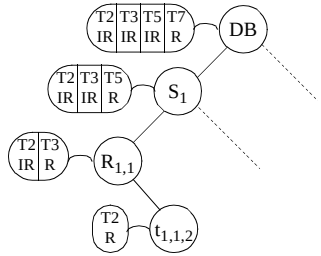
g) Es seien zwei Arten von speziellen Anwartschaftssperren IR und IX vorhanden. Wie ändern sich die Sperrprotokolle?

h) Berechnen Sie jeweils die Anzahl der Sperranforderungen und der Deaktivierungen, die insgesamt verbrauchten Zeiteinheiten sowie die durchschnittlichen Wartezeiteinheiten.

g) + h): **Hierarchische Sperrprotokolle mit speziellen Anwartschaftssperren.**



Zum Zeitpunkt Z_{16} ergibt sich folgende Situation:



Anzahl der Sperranforderungen: Insgesamt 19 Sperren angefordert

Anzahl der Deaktivierungen : Insgesamt 3 Transaktionen in Wartestellung

Transaktion	Dauer	Zeiteinheiten	Wartezeiten	Bemerkung
T_1	Z_1 - Z_{11}	10	0	
T_2	Z_2 - Z_{22}	20	0	
T_3	Z_3 - Z_{31}	28	8	„läuft“ erst nach Ende T_1
T_4	Z_4 - Z_{14}	10	0	
T_5	Z_5 - Z_{34}	29	9	„läuft“ erst nach Ende T_4
T_6	Z_6 - Z_{16}	10	0	
T_7	Z_7 - Z_{36}	29	9	„läuft“ erst nach Ende T_6

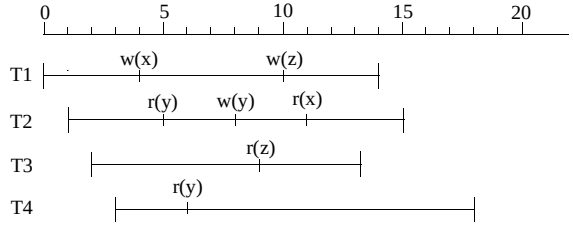
Gesamtdauer: 36 Zeiteinheiten

Durchschnittliche Wartezeit:

$$W = \frac{8 + 9 + 9}{7} = \frac{26}{7} = 3,7$$

Aufgabe 3: Vergleich verschiedener Synchronisationsverfahren

Gegeben ist der Ablaufplan von vier Transaktionen T1-T4:



a) Ist der Ablaufplan serialisierbar? Welche seriellen Historien ergeben sich?

b) Geben Sie die Serialisierungsreihenfolge an, wenn zur Synchonisation verwendet wird:

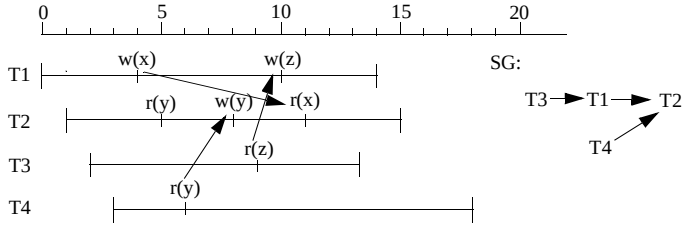
(1) RX
(2) RUX (symmetrisch)
(3) RUX (unsymmetrisch)
(4) RAX
(5) RAC
(6) BOCC
(7) BOCC+ (mit Zeitstempeloptimierung)
(8) FOCC

Gehen Sie von starkem 2-Phasen-Locking aus (Halten aller Sperren bis zum EOT, dann atomares Freigeben).

Hinweis: Die vom obigen Ablaufplan vorgegebenen Zeitabstände zwischen den einzelnen Operationen innerhalb der individuellen Transaktionen sollen auch im Falle von Blockierungen durch Sperranforderungen beibehalten werden. Sie sind dann als Zeit zwischen der tatsächlichen Ausführung der ersten Operation (d.h. nach der Gewährung einer Sperre) und der (versuchten) Ausführung der zweiten Operation zu verstehen.

Lösung:

a) Ist der Ablaufplan serialisierbar? Welche seriellen Historien ergeben sich?



Einzeichnen der Konfliktoperationen ergibt folgenden Serialisierbarkeitsgraphen:

Der resultierende Serialisierbarkeitsgraph ist azyklisch, der Ablaufplan daher serialisierbar. Mögliche serielle Ablaufpläne sind:

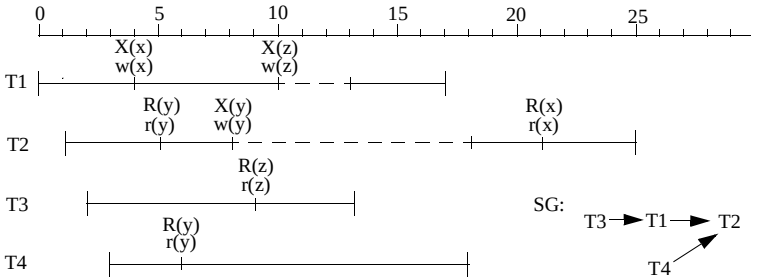
T3|T1|T4|T2

T3|T4|T1|T2

T4|T3|T1|T2

b) Geben Sie die Serialisierungsreihenfolge an, wenn zur Synchronisation folgende Verfahren verwendet werden:

(1) RX



t=4: T1 schreibt x und erhält dafür die nötige X-Sperre.

t=5: T2 liest y und erhält R-Sperre.

t=6: T4 liest ebenfalls y und erhält ebenfalls eine R-Sperre (R ist mit R verträglich)

t=8: T2 versucht y zu schreiben. Die dazu erforderliche Sperrkonversion von der R- zur X-Sperre führt zur Blockierung, da T4 eine R-Sperre hält.

t=9: T3 liest z und erhält R-Sperre.

t=10: T1 versucht z zu ändern, die dazu nötige X-Sperre kann zunächst nicht gewährt, werden da T3 parallel z liest, T1 muss warten.

t=13: EOT von T3, Freigabe der R-Sperre auf z, folglich kann T1 deblockiert werden und erhält die X-Sperre.

t=17: EOT von T1, Freigabe der X-Sperren auf x und z.

t=18: EOT von T4, Freigabe der R-Sperre auf y, T2 kann deblockiert werden und erhält die X-Sperre auf y.

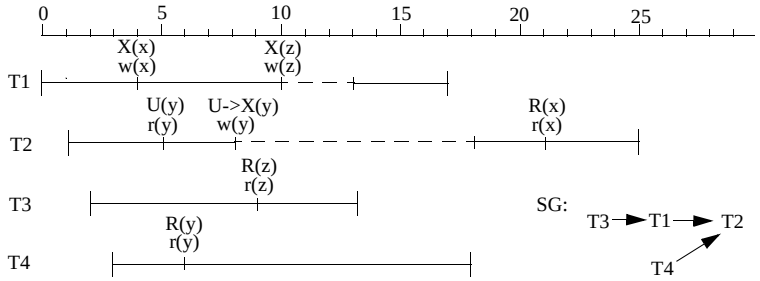
t=21: T2 liest x und erhält dafür eine R-Sperre.

t=25: EOT von T2, Freigabe aller Sperren.

Im Beispielszenario wird ein Problem des RX-Verfahrens nicht deutlich, der Konversions-Deadlock. Diese würde auftreten, wenn z.B. T4 zum Zeitpunkt t=15 eine Sperrkonversion des R(y) auf X(y) probieren würde. In diesem Fall würde T2 auf die Freigabe des R von T4 warten und umgekehrt..

	Dauer	Wartezeit
T1	17	3
T2	24	10
T3	11	0
T4	15	0
Gesamt	25	AVG 3,25

(2) RUX (symmetrisch)



t=4: T1 schreibt x und erhält dafür die nötige X-Sperre.

t=5: T2 liest y mit Änderungsabsicht und erhält eine U-Sperre.

t=6: T4 liest ebenfalls y und erhält eine R-Sperre (R ist beim symmetrischen RUX mit R verträglich)

t=8: T2 versucht y zu schreiben. Die dazu erforderliche Sperrkonversion von der U- zur X-Sperre führt zur Blockierung, da T4 eine R-Sperre hält.

t=9: T3 liest z und erhält R-Sperre.

t=10: T1 versucht z zu ändern, die dazu nötige X-Sperre kann zunächst nicht gewährt, werden da T3 parallel z liest, T1 muss warten.

t=13: EOT von T3, Freigabe der R-Sperre auf z, folglich kann T1 deblockiert werden und erhält die X-Sperre.

t=17: EOT von T1, Freigabe der X-Sperren auf x und z.

t=18: EOT von T4, Freigabe der R-Sperre auf y, T2 kann deblockiert werden und erhält die X-Sperre auf y.

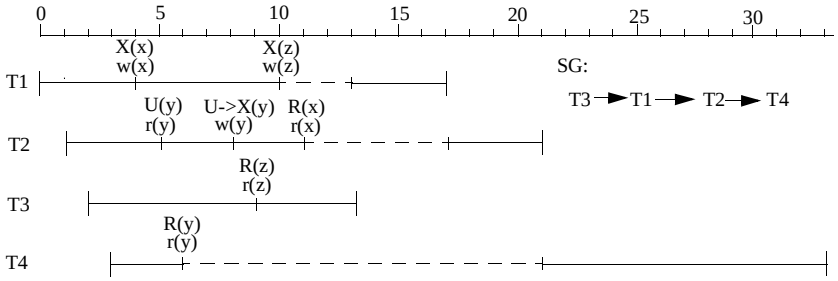
t=21: T2 liest x und erhält dafür eine R-Sperre.

t=25: EOT von T2, Freigabe aller Sperren.

Der resultierende Ablaufplan ist für dieses Beispiel identisch zu dem von RX. Der bei der Erläuterung von RX geschilderte Fall des Konversions-Deadlocks kann bei RUX nicht eintreten. Hier hätte T4 zum Zeitpunkt t=6 eine U-Sperre angefordert, und wäre daraufhin blockiert worden (U mit gesetztem U unverträglich).

Im symmetrischen RUX-Verfahren könnten während der Wartezeit von T2 auf das Ende T4 beliebige neue Leser hinzukommen (R ist mit gesetztem U verträglich) und so die Bearbeitung von T2 unbegrenzt verzögern (Starvation).

(3) RUX (unsymmetrisch)



t=4: T1 schreibt x und erhält die X-Sperre.

t=5: T2 liest y mit Änderungsabsicht und erhält U-Sperre.

t=6: T4 versucht, y zu lesen. R ist jetzt jedoch unverträglich mit gesetztem U, daher muss T4 warten.

t=8: T2 schreibt y, die dazu erforderliche Sperrkonversion von der U- zur X-Sperre ist nun möglich, da kein paralleler Leser eine R-Sperre hält (es könnten jedoch noch Lesesperren von älteren Lesern gesetzt sein, die ihr R vor dem U von T2 angefordert haben).

t=9: T3 liest z und erhält R-Sperre.

t=10: T1 versucht z zu ändern, die dazu nötige X-Sperre kann zunächst nicht gewährt, werden da T3 parallel z liest, T1 muss warten.

t=11: T2 versucht x zu lesen, allerdings ist x noch von T1 mit X gesperrt, T2 muss warten.

t=13: EOT von T3, Freigabe der R-Sperre auf z, folglich kann T1 deblockiert werden und erhält die X-Sperre.

t=17: EOT von T1, Freigabe der X-Sperren auf x und z. T2 kann deblockiert werden und erhält R(x).

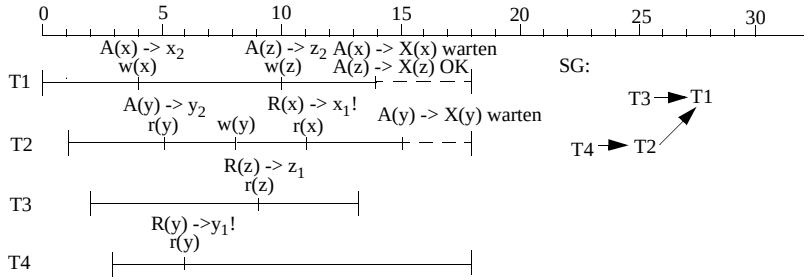
t=21: EOT von T2, Freigabe von X(y) und R(x), T4 kann deblockiert werden und erhält R(y).

t=33: EOT von T4, Freigabe von R(y).

Die Gesamtdauer des Ablaufplans und die durchschnittliche Wartezeit der Transaktionen verschlechtern sich im gezeigten Szenario, allerdings wird das Aushungern von Schreibern durch neu hinzukommende Leser verhindert und so die "Fairness" verbessert.

	Dauer	Wartezeit
T1	17	3
T2	21	6
T3	11	0
T4	30	15
Gesamt	33	AVG 6

(4) RAX



Alle Transaktionen fordern nun vor dem Schreiben (bzw. vor dem Lesen mit Änderungsabsicht) eine A-Sperre an und erhalten (wenn die Sperre gewährt wird) eine neue Version des Objekts, auf der sie arbeiten können. Während der Bearbeitung dieser Kopie sehen neue Leser die alte Version.

t=4: T1 ändert x und erhält so eine neue Version x_2.

t=5: T2 liest y mit Änderungsabsicht und erhält ebenfalls eine private Version.

t=6: T4 liest y und erhält die alte Version y_1.

t=8: T2 ändert ihre private Version.

t=9: T3 liest z und sieht die (zu diesem Zeitpunkt einzige) Version z_1.

t=10: T1 ändert z und erstellt somit eine neue Version z_2.

t=11: T2 liest x und sieht die alte Version x_1, da die Änderung von T1 noch nicht freigegeben wurde.

t=13: EOT von T3, Freigabe der R-Sperre auf z.

t=14: EOT von T1, Konversion aller A-Sperren nach X zum Einbringen der geänderten Version. Konversion von A(z) nach X(z) gelingt (da keine Leser die alte Version z_1 noch benötigen). Konversion von A(x) nach X(x) blockiert, da der "alte" Leser T2 noch auf der alten Version x_1 arbeitet.

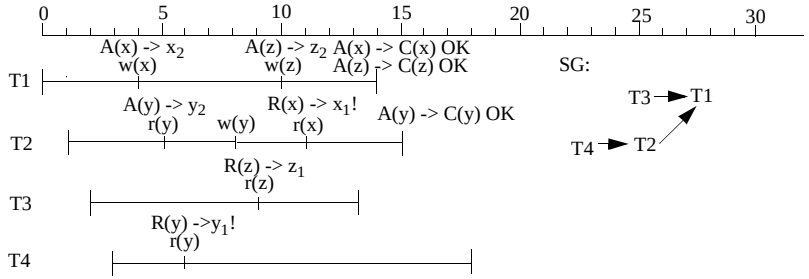
t=15: EOT von T2, Konversion von A(y) nach X(y) blockiert, da der alte Leser T4 noch auf der alten Version y_1 arbeitet.

t=18: EOT von T4 gibt Lesesperre auf y frei. T2 kann seine geänderte Version y_2 freigeben, y_1 wird nicht mehr benötigt und kann verworfen werden. T2 gibt bei seinem erfolgreichen EOT die Lesesperre auf x frei, daraufhin kann T1 seine geänderte Version x_2 einbringen und abgeschlossen werden.

Die resultierende Schedule hat eine kürzere Gesamtdauer und eine geringerer durchschnittliche Wartezeit als RX und die beiden Varianten von RUX. Allerdings ist RAX nicht chronologieerhaltend, da hier T2 im äquivalenten seriellen Ablaufplan vor T1 ausgeführt würde.

	Dauer	Wartezeit
T1	18	4
T2	17	3
T3	11	0
T4	15	0
Gesamt	18	AVG 1,75

(5) RAC



t=4 bis t=13: wie RAX

t=14: EOT von T1, Konversion der A-Sperren nach C-Sperren für jedes geänderte Objekt. Keine Konflikte bei der Konversion von A(x) nach C(x), da die alte Version x_1 für T2 noch bereitgehalten wird. Nach erfolgtem Einbringen der geänderten Versionen von x und z erfolgt eine Freigabe der C-Sperren sobald alle alten Leser fertig sind (im Falle von z also unmittelbar, für x nach dem Ende von T2).

t=15: EOT von T2, Konversion von A(y) nach C(y) erfolgreich, y_1 wird noch für T4 bereitgehalten. Freigabe des C(x), das von T1 hinterlassen wurde. Beibehalten des C(y) bis zum Ende von T4.

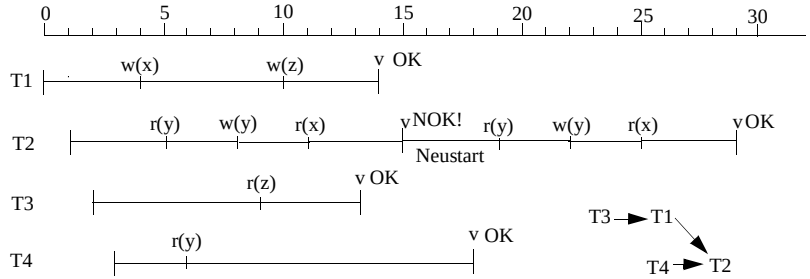
t=18: EOT von T4. Freigabe von R(y) und damit auch des C(y) von T2.

Im Vergleich zu RAX werden Schreiber bei ihrem Commit nicht blockiert, da bei RAC zwei gültige Versionen gleichzeitig *zum Lesen* bereitgehalten werden können (erkennbar an gesetztem C). Die alte Version wird für alle Leser bereitgehalten, die vor dem EOT des Schreibers eine Lesesperre angefordert haben, die neue Version ist für alle neuen Leser sichtbar. Wenn die letzten "alten" Leser fertig sind, kann C entfernt werden. Erst jetzt kann ein neuer Schreiber eine neue Version mit A anlegen.

Der resultierende Schedule sieht auf den ersten Blick aus wie der in der Aufgabenstellung vorgegebene. Allerdings ist hier wie auch bei RAX durch die Versionierung die Reihenfolge von T1 und T2 in den äquivalenten seriellen Abläufen invertiert.

	Dauer	Wartezeit
T1	14	0
T2	14	0
T3	11	0
T4	15	0
Gesamt	18	AVG 0

(6) BOCC



Bei BOCC validiert jede TA bei EOT gegen alle Schreiber, die während ihrer Laufzeit erfolgreich validiert (und somit ihre Änderungen festgeschrieben) haben. Dabei wird die Schnittmenge der von der validierenden Transaktion gelesenen Objekte (ihr Readset) mit der Menge aller von den erfolgreich validierten TAs geschriebenen Objekte (Vereinigung der Writesets) verglichen. Wenn diese Schnittmenge nicht leer ist, wird dies als Konflikt erkannt und die Transaktion zurückgesetzt. Andernfalls werden die Änderungen der Transaktion, die zuvor in einem privaten Puffer durchgeführt wurden, in die Datenbank eingebracht (Schreibphase).

Validierung T1: keine vorher validierenden Schreiber, kein Konflikt

Validierung T2: $RS_2 = \{x,y\}$, $WS = WS_1 = \{x\}$ $RS_2 \cap WS_1 = \{x\} \neq \emptyset$ => Konflikt, verwerfen der Änderungen und Wiederanlauf

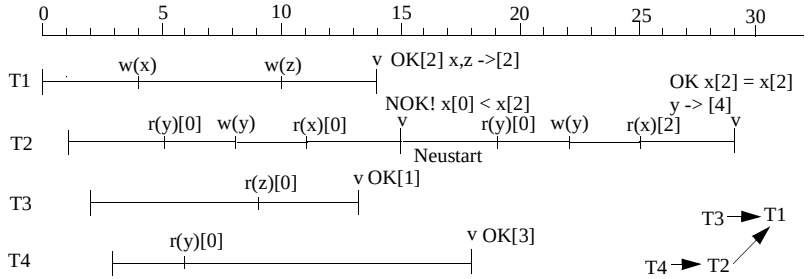
Validierung T3: keine vorher validierenden Schreiber, kein Konflikt

Validierung T4: Schreiber von y T2 validiert nicht erfolgreich, daher kein Konflikt

2. Validierung von T2: kein parallel validierender Schreiber auf x oder y, kein Konflikt.

	Dauer	Wartezeit
T1	14	0
T2	28	(14)
T3	11	0
T4	15	0
Gesamt	28	AVG 3,5

(7) BOCC+ (mit Zeitstempeloptimierung)



BOCC+ verzichtet auf das führen von Writesets. Stattdessen verteilt es aufsteigende Zeitstempel für erfolgreich validierende TA. Alle von der TA geschriebenen Objekte werden mit diesem Zeitstempel versehen. Beim Lesen eines Objekts wird dessen Zeitstempel im Readset der lesenden TA vermerkt. Bei Validierung wird für jedes gelesene Objekt überprüft, ob der zum Validierungszeitpunkt aktuelle Zeitstempel gleich dem Zeitstempel zum Lesezeitpunkt ist. Ist der Lesezeitstempel kleiner, hat zwischenzeitlich eine andere TA das Objekt geändert und erfolgreich validiert, die Transaktion muss abgebrochen und ggf. wiederholt werden.

Validierung T1: keine Leseoperationen, kein Konflikt

Validierung T2: T2 liest y mit Zeitstempel 0 und vermerkt dies in seinem Readset. Vor der Validierung von T2 validiert T1 und erhöht dabei den Zeitstempel von y auf 1. Bei der Validierung von T2 wird erkannt, dass der nun aktuelle Zustand von y jünger als der von T2 gelesene ist, es erfolgt ein Wiederanlauf.

Validierung T3: Zeitstempel von z zum Lese- und Validierungszeitpunkt sind identisch, kein Konflikt. z wurde zwar bereits von T1 geändert (in dessen privatem Puffer), aber noch nicht eingebracht.

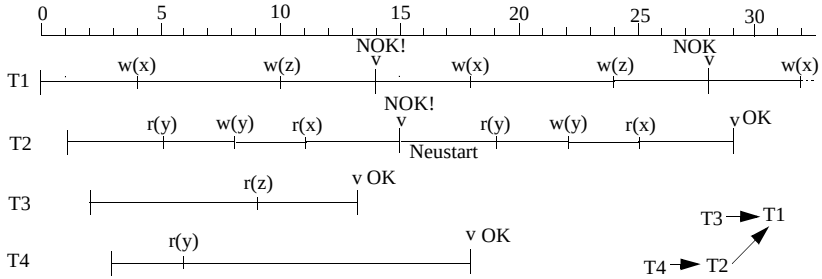
Validierung T4: Zeitstempel von y zum Lese- und Validierungszeitpunkt sind identisch (da T2 nicht erfolgreich validiert hat und daher auch der Zeitstempel von y nicht erhöht wurde), kein Konflikt.

2. Validierung T2: Zeitstempel von z zum Lese- und Validierungszeitpunkt sind identisch, kein Konflikt

Gesamtdauer und Wartezeiten wie BOCC.

BOCC+ bringt einen Vorteil gegenüber BOCC, wenn eine Transaktion ein Objekt einer während ihrer Laufzeit validierenden Schreib-TA liest, *nachdem* die Schreib-TA validiert hat. Hier würde das ungenauere BOCC einen Konflikt erkennen, obwohl das korrekte Objekt gelesen wurde.

(8) FOCC



FOCC vergleicht das Writeset einer validieren TA mit den Readsets aller zum Validierungszeitpunkt aktiven TAs. Ist die Schnittmenge nicht leer, wurde ein Konflikt erkannt. Im Gegensatz zu BOCC kann hier die zurückzusetzende TA frei gewählt werden. Dadurch kann der Verlust an bereits getaner Arbeit minimiert oder bereits einmal wegen eines Konflikts wiederholte TAs können bevorzugt werden. Im obigen Beispiel wird von einem naiven TA-Manager grundsätzlich die validierende TA zurückgesetzt, was in der Praxis nicht die optimale Strategie wäre.

Validierung T1: $WS_1 = \{x,z\}$, $RS_2 = \{x,y\}$, $RS_4 = \{y\}$, $RS_2 \cup RS_4 = RS = \{x, y\}$ $WS_1 \cap RS = \{x\} \neq \emptyset$ => Konflikt, Neustart von T1

Validierung T2: $WS_2 = \{y\}$, $RS_1 = \{\}$, $RS_4 = \{y\}$, $RS_1 \cup RS_4 = RS = \{y\}$, $WS_2 \cap RS = \{y\} \neq \emptyset$ => Konflikt, Neustart von T2

Validierung T3: leeres Writeset, kein Konflikt

Validierung T4: leeres Writeset, kein Konflikt

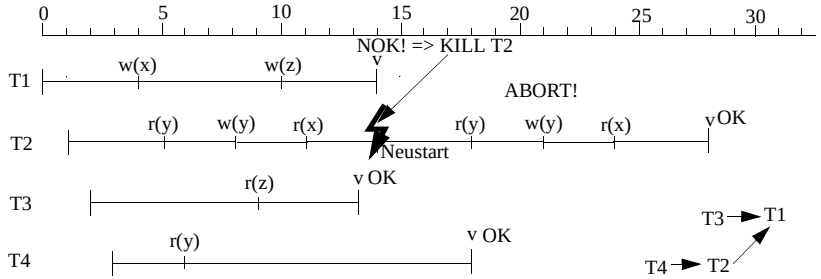
2. Validierung T1: $WS_1 = \{x,z\}$, $RS_2 = RS = \{x,y\}$, $WS_1 \cap RS = \{x\} \neq \emptyset$ => Konflikt, 2. Neustart von T1!

2. Validierung T2: $WS_2 = \{y\}$, $RS_1 = RS = \{\}$, $WS_2 \cap RS = \emptyset$ => kein Konflikt

3. Validierung T1: keine parallelen TA, kein Konflikt

	Dauer	Wartezeit
T1	42	(28)
T2	28	(14)
T3	11	0
T4	15	0
Gesamt	42	AVG 10,5

FOCC mit einem "intelligenteren" Transaktionsmanager:



Validierung T1: $WS_1 = \{x,z\}$, $RS_2 = \{x,y\}$, $RS_4 = \{y\}$, $RS_2 \cup RS_4 = RS = \{x, y\}$
$WS_1 \cap RS = \{x\} \neq \emptyset$ => Konflikt, Abbruch und Neustart der "anderen" TA T2

Validierung T3: leeres Writeset, kein Konflikt

Validierung T4: leeres Writeset, kein Konflikt

Validierung T2: keine parallelen TA zum Validierungszeitpunkt, kein Konflikt

	Dauer	Wartezeit
T1	14	0
T2	28	(14)
T3	11	0
T4	15	0
Gesamt	28	AVG 3,5