Zeitstempelverfahren

Udo Kelter

02.06.2001

Zusammenfassung dieses Lehrmoduls

Zeitstempelverfahren sind sog. validierende Concurrency-Control-Verfahren, bei denen inkorrekte Verzahungen von Transaktionsausführungen verhindert werden, indem abhängig von Validationstests einzelne Transaktionen zurückgesetzt und neugestartet werden. Die Validationstests werten die Zeitpunkte von Zugriffen zu Objekten aus. Dieses Lehrmodul erläutert zunächst das Validationsprinzip und seine Vor- und Nachteile. Weiter wird die Grundform der Zeitstempelverfahren vorgestellt, ferner Techniken zur Erzeugung und Verwaltung von Zeitstempeln. Eine Vereinigung der Vorteile von Sperr- und Zeitstempelverfahren ist in einigen kombinierten Verfahren gelungen, die anschließend vorgestellt werden.

Vorausgesetzte Lehrmodule:

obligatorisch:
- Transaktionen und die Integrität von Datenbanken
- Sperrverfahren

empfohlen:
- Recovery

Stoffumfang in Vorlesungsdoppelstunden: 1.0

Inhaltsverzeichnis

1 Validierende Verfahren

1.1 Verhinderung inkorrekter Verzahnungen durch Neustart

Bei Sperrverfahren werden inkorrekte Verzahnungen dadurch verhindert, daß einzelne Zugriffe von Transaktionen verzögert werden. Technisches Mittel zum Verzögern waren Sperren, auf deren Freigabe eine Transaktion ggf. warten muß. Infolge des Wartens können Deadlocks auftreten. Deadlocks können i.a. nicht durch präventive Maßnahmen verhindert, sondern nur entdeckt und aufgelöst werden (s. [SPV]). Für diesen Zweck sind zum einen Software-Komponenten innerhalb der Concurrency-Control-Komponente vorzusehen, ferner Datenstrukturen, die eine effiziente Deadlock-Erkennung ermöglichen (Wartegraph). Außerdem müssen laufend vorbereitende Maßnahmen zur Deadlock-Erkennung durchgeführt und im Falle eines Deadlocks gewisse Transaktionen zurückgesetzt werden.

Die Maßnahmen zur Deadlock-Behandlung sind schon bei zentralen Datenbanken aufwendig und natürlich unerwünscht. Bei verteilten Datenbanken ist schon die Deadlock-Erkennung äußerst aufwendig: Der Wartezyklus kann sich über verschiedene Rechner erstrecken, so daß zu seiner Erkennung systemweite (und sehr teure) Kommunikation erforderlich wird. Gesucht sind also deadlockfreie Verfahren, bei denen die Transaktionen nicht vorab deklarieren müssen, zu welchen Objekten sie insgesamt zugreifen werden. Dies ist die Hauptmotivation für sog. **validierende CC-Verfahren**.

Die Grundidee dieser Verfahren besteht darin, überhaupt nicht zu warten, sondern inkorrekte Verzahnungen durch Rücksetzen einzelner Transaktionen zu verhindern. Diese Verfahren arbeiten nach folgendem Prinzip:

- Alle Zugriffe werden, wenn überhaupt, dann sofort ausgeführt.
- Bei bestimmten Gelegenheiten, z.B. bei jedem Zugriff oder am Ende jeder Transaktion, werden **Validationstests** durchgeführt: es wird überprüft, ob die bisherige effektiv eingetretene Verzahnung korrekt

(also mindestens cp-serialisierbar) ist. Falls nicht, wird die Transaktion zurückgesetzt und automatisch neugestartet. Dies bezeichnen wir in diesem Kapitel kurz mit **Neustart**. Der letzte (und oft einzige) Validationstest findet im Rahmen des Commits statt. Führt er nicht zum Neustart, ist die Transaktion insgesamt **validiert**. (Üblich ist auch die Bezeichnung "zertifiziert".)

Durch dieses Prinzip wird natürlich die Häufigkeit von Rollback merklich erhöht. In der Konsequenz ist es dringend geboten, Fortpflanzung von Rollback zu verhindern. Daher nehmen wir bei allen folgenden Verfahren das verzögerte Schreiben (*deferred update*) an[1], wobei veränderte Objekte erst bei Commit in die Datenbank geschrieben und vorher gepuffert werden. Aus diesem Grund gibt es keine ungesicherten Werte in der Datenbank, alle geschriebenen Werte sind von validierten Transaktionen geschrieben worden.

1.2 Vergleich mit Sperrverfahren

Im Vergleich zum Sperren hat das Validieren folgende Vor- und Nachteile:

- Die validierenden Verfahren sind deadlockfrei. Weder Erkennungsmechanismen (Software) noch vorbereitende Maßnahmen zur Deadlock-Erkennung sind erforderlich. Ebenso entfallen Kosten der Deadlock-Auflösung.
- Die Vermehrung des Rollbacks von Transaktionen ist – unabhängig von der vereinfachten Realisierung des Rollbacks durch das verzögerte Schreiben – ein Nachteil.
- Zur Realisierung des verzögerten Schreibens müssen lokale Kopien der Objekte vorgesehen werden, bei der Realisierung durch Sperren ist dies nicht erforderlich. Validierende Verfahren sind daher wenig für Anwendungsfälle geeignet, bei denen Transaktionen viele Objekte schreiben.

[1] S. auch Ausführungen zum verzögerten Schreiben in Abschnitt 4.4 in [REC].

- Wenn eine Transaktion abgebrochen und neugestartet worden ist, kann ihr dieses "Pech" beim nächsten Ausführungsversuch erneut widerfahren. Frühere Neustarts einer Transaktion verbessern die Wahrscheinlichkeit nicht, daß diese Transaktion bei ihrem nächsten Ausführungsversuch nicht noch einmal neugestartet wird. Bei den Grundformen der validierenden Verfahren kann (zumindest theoretisch) ein zyklischer Neustart eintreten, d.h. eine Transaktion wird endlos neugestartet. Betroffen sind vor allem längere Transaktionen, denn die Wahrscheinlichkeit eines Neustarts steigt mit der Länge einer Transaktion. Alle Maßnahmen zur Lösung des Neustartproblems beruhen in irgendeiner Weise auf Sperren, allerdings so, daß keine Deadlocks möglich sind. Die Verfahren sind dann allerdings wieder in Sperren involviert, es müssen Mechanismen zur Realisation der Sperren vorgesehen werden, und die Verfahren werden relativ kompliziert.
- Bezüglich des Aufwands für die Hilfsdaten ist keine pauschale Aussage zugunsten von Neustarts oder Sperrungen möglich. Bei validierenden Verfahren entfällt zwar die Sperrtabelle (sofern Sperren zur Lösung des Neustart-Problems nicht doch wieder eingeführt werden), dafür sind aber andere Hilfsdaten für die Validationstests erforderlich, die bzgl. Speicherplatzbedarf und Handhabungsaufwand durchaus mit der Sperrtabelle vergleichbar sein können.

Eine quantitative Abwägung der Vor- und Nachteile einzelner Verfahren (z.B. mit stochastischen Modellen) ist wegen der hohen Komplexität und Vielfalt der Einflußfaktoren sehr schwierig und gleichzeitig problematisch. Das CC-Verfahren ist nur einer von mehreren wesentlichen Einflußfaktoren, die die Gesamtleistung eines DBS bestimmen. Wir beschränken uns deshalb im folgenden auf einen qualitativen Vergleich wichtiger Einzelaspekte der Verfahren; es sei noch einmal davor gewarnt, von der Überlegenheit eines Verfahrens in Einzelaspekten auf eine generelle Überlegenheit unter beliebigen Einsatzbedingungen zu schließen (einige triviale Fälle einmal ausgenommen). Für eine Auswahl eines Verfahrens in einem konkreten Fall sind darüber hinaus noch andere Aspekte wesentlich, z.B. die Komplexität bzw. der

Realisierungsaufwand für die Software.

1.3 Varianten validierender Verfahren

Das Grundprinzip des Validierens läßt Raum für viele Varianten. Die beiden wichtigsten Arten von validierenden CC-Verfahren sind:

- Zeitstempel-Verfahren und
- optimistische Verfahren.

Die Verfahren unterscheiden sich vor allem in folgenden Punkten:

***Zeitpunkt der Validationstests*:** Wann werden Validationstests durchgeführt?

***Korrektheitsbegriff*:** Welche eingetretenen Verzahnungen werden als korrekt erachtet und welche nicht?

***Hilfsdaten*:** Welche Hilfsdaten werden für die Entscheidung benutzt?

Alle drei Aspekte hängen natürlich zusammen; wir wollen jedoch schon vorab einige Alternativen isoliert besprechen.

Zum Zeitpunkt der Validationstests: Sinnvolle Gelegenheiten für Validationstests sind neben dem Commit, das alle Schreib-Aktionen enthält, nur einzelne frühere Lese-Aktionen. Zeitstempel-Verfahren führen bei jedem Zugriff einen Test durch, optimistische Verfahren nur bei Commit.

Für die frühen Tests bei Leseaktionen spricht, daß inkorrekte effektive Verzahnungen früher erkannt werden können. Durch das Rücksetzen der Transaktion geht dann weniger geleistete Arbeit verloren, die Tests sind relativ einfach, allerdings häufig.

Bei späten Tests kann theoretisch das gesamte Geschehen im Verlauf der Transaktionsausführung berücksichtigt werden, es können also unnötige Neustarts und viele Einzeltests vermieden werden. Jedoch steigt die Komplexität der Tests dann ganz erheblich, so daß der Testaufwand eher größer ist als bei frühen Tests.

Zum Korrektheitsbegriff und den Hilfsdaten: In [SPV] hatten wir eine Verzahnung als serialisierbar definiert, wenn es zu jeder enthaltenen Transaktion einen Serialisierungspunkt gibt, d.h. alle Ereignisse einer Transaktion können konfliktfrei zu ihrem Serialisierungspunkt verschoben werden. Dieser Korrektheitsbegriff ist allerdings für die Verhältnisse, die bei validierenden Verfahren unterstellt werden, aus Aufwandsgründen weniger geeignet.

Hierzu müssen wir zunächst die Korrektheitsbegriffe für Verzahnungen etwas näher betrachten. Serialisierungspunkte waren so motiviert, daß hier scheinbar die gesamte Transaktion stattfinden kann. Wenn wir alle Aktionen dementsprechend verschieben, erhalten wir einen **seriellen** Ablauf. Die aufgetretene Verzahnung und dieser serielle Ablauf sind dann **äquivalent** in dem Sinne, daß sie beide die gleiche Menge von Aktionen umfassen und daß, dann, wenn zwei Aktionen in Konflikt miteinander stehen, diese beiden Aktionen in beiden Abläufen in der gleichen Reihenfolge auftreten, also nicht vertauscht werden. Die vorstehende Definition der Äquivalenz zweier Abläufe wird auch **cp-Äquivalenz** (*conflict-preserving equivalence*) genannt. Es gibt noch andere Definitionen der Äquivalenz von Abläufen, auf die wir hier aber nicht eingehen.

Unserer bisherige Definition von Serialisierbarkeit ist mit diesen Begriffen äquivalent zu der folgenden: Eine Verzahnung ist **serialisierbar**, wenn es einem cp-äquivalenten seriellen Ablauf gibt.

Theoretisch kann für jeden Korrektheitsbegriff ein validierendes Verfahren entwickelt werden: Die bis zum Testzeitpunkt abgelaufene Verzahnung muß in ihren relevanten Details bekannt sein, d.h. entsprechende Daten sind laufend zu speichern; getestet wird bei jeder Aktion oder am Ende einer Transaktion, ob die entstandene Verzahnung Präfix einer korrekten Verzahnung ist, wobei mit dem Rücksetzen aller noch nicht beendeten Transaktionen gerechnet werden muß. Fällt der Test negativ aus, wird die Transaktion, die den Test verursachte, zurückgesetzt und automatisch neugestartet.

Wegen der Häufigkeit, mit der die Tests durchgeführt werden, ist

deren Effizienz sehr kritisch[2]. Wenn man nun entscheiden will, ob ein gegebener Ablauf serialisierbar ist, muß man prüfen, ob ein cp-äquivalenter serieller Ablauf existiert oder nicht. Hierfür sind keine ausreichend effizienten Verfahren verfügbar.

Bei den praktisch brauchbaren Verfahren wird ein Trick angewandt, durch den die Korrektheitsüberprüfung stark vereinfacht wird. Für die Serialisierbarkeit reicht es aus, wenn es *irgendeinen* seriellen Ablauf gibt, der äquivalent zur vorhandenen Verzahnung ist. Die Vereinfachung besteht darin, *einen ganz bestimmten* seriellen Ablauf vorzugeben. Vorgegeben wird natürlich ein serieller Ablauf, der mit möglichst geringer Wahrscheinlichkeit zu einem Neustart führt. In etwa wird dieser serielle Ablauf gemäß den Ankunftszeitpunkten der Transaktionen gebildet.

Scheduling. Eine Umformung einer Aufrufsequenz von Aktionen in eine effektive Ausführungssequenz (s. Abschnitt 2) findet bei validierenden Verfahren nicht in dem Sinne wie bei Sperrverfahren statt: Die Aufrufsequenz wird durch das Rücksetzen selbst verändert. Sofern keine Transaktion zurückgesetzt wird, ist die Aufrufsequenz gleichzeitig effektive Ausführungssequenz. Unter der **effektiven Ausführungssequenz** verstehen wir daher in diesem Lehrmodul die letztlich wirksame Aufrufsequenz. Die Umformung der "ursprünglichen" Aufrufsequenz in die effektive Ausführungssequenz ist nicht reproduzibel, denn die Zeit bis zum Neustart einer Transaktion ist zufällig.

2 Zeitstempel-Verfahren

2.1 Die Grundform

Bei der Grundform der Zeitstempel-Verfahren wird bei jedem Zugriff ein Validationstest durchgeführt. Getestet wird, ob die bisherige effektive Ausführungssequenz cp-äquivalent zu dem seriellen Ablauf ist, der sich aus der Ankunftszeit der Transaktionen ergibt. Für zurückgesetzte Transaktionen ist der Zeitpunkt ihres Neustarts relevant.

[2] Anmerkung: bei Sperrverfahren werden solche Tests überhaupt nicht durchgeführt, die entstehenden Verzahnungen sind automatisch serialisierbar.

Das wichtigste technische Hilfsmittel sind **Zeitstempel**. Jede Transaktion T_i erhält von der CC-Komponente einen eigenen, eindeutigen Zeitstempel, $\mathbf{Z(T_i)}$. Den Zeitstempel erhält die Transaktion bei ihrer Ankunft oder später, spätestens vor ihrem ersten lesenden Zugriff. Einen Zeitstempel stellen wir uns vorerst am besten als die aktuelle Uhrzeit, ggf. verbunden mit dem Datum, vor, und zwar in einer Genauigkeit, daß keine zwei Transaktionen den gleichen Zeitstempel und eine spätere Transaktion einen "größeren" Zeitstempel erhalten. Wir werden "Zeitstempel" und "Zeitpunkt" als Synonyme benutzen.

Der Zeitstempel einer Transaktion ist der vorgegebene Serialisierungspunkt dieser Transaktion in der entstehenden effektiven Ausführungssequenz. Alle Ereignisse in der effektiven Ausführungssequenz müssen also konfliktfrei zu den jeweiligen Zeitstempeln der Transaktionen verschoben werden können. Diese Verschiebung ist nur dann nicht möglich, wenn folgende typische Situation vorliegt:

```
Ti Z------------ei
Tj        Z-----ej
```

In diesem und den folgenden Beispielen benutzen wir eine graphische Notation für Abläufe, in der das Symbol 'Z' den Zeitstempel darstellt, `r(x)` und `w(x)` das Lesen bzw. Schreiben eines Objekts x.

Im obigen Beispiel seien e_i und e_j zwei Ereignisse, die in Konflikt stehen. e_i kann nicht konfliktfrei zum Zeitstempel von T_i verschoben werden, denn e_j müßte hierfür vor den Zeitstempel von T_j verschoben werden.

Zwischen e_i und e_j können drei verschiedene Arten von Konflikten bzgl. eines Objekts x bestehen:

Lese-Schreib-Konflikt:

```
Ti Z--------r(x) ....
Tj        Z--w(x)
```

Schreib-Lese-Konflikt:

```
Ti Z--------w(x)
Tj        Z--r(x)
```

Schreib-Schreib-Konflikt:

```
Ti Z--------w(x)
Tj        Z--w(x)
```

Da wir verzögertes Schreiben annehmen, kann nach einem Schreibereignis einer Transaktion kein weiteres Ereignis von dieser Transaktion mehr stattfinden. Bei Lese-Ereignissen können jedoch noch weitere Ereignisse geplant sein, was durch "..." dargestellt ist.

Bei jedem Zugriff e_i einer Transaktion T_i muß folgender Validationstest durchgeführt werden: T_i ist abzubrechen, wenn es eine *jüngere* Transaktion T_j (d.h. der Zeitstempel von T_j ist größerer als der von T_i) gibt und wenn (a) T_j schon *vor* e_i einen Zugriff e_j durchgeführt hat und (b) e_j in *Konflikt* mit e_i steht. Andernfalls kann der Zugriff ausgeführt werden.

Zur Vereinfachung der Suche nach einem derartigen T_j sehen wir bei jedem Objekt zwei Zeitstempel vor:

`ZR(x):` Zeitstempel der jüngsten Transaktion, die x *gelesen* hat (m.a.W. gibt es ein Leseereignis, das nicht vor diesen Zeitpunkt verschoben werden kann).

`ZW(x):` Zeitstempel der jüngsten Transaktion, die x *geschrieben* hat (m.a.W. gibt es ein Schreibereignis, das nicht vor diesen Zeitpunkt verschoben werden kann).

Bei jedem erfolgreichen Zugriff müssen die Hilfsvariablen `ZR` und `ZW` auf den neuen Stand gebracht werden. Diese Aktualisierung führen wir zusammen mit dem Validationstest aus. Insgesamt erhalten wir folgende Testalgorithmen:

- T_i will x lesen (Test auf Lese-Schreib-Konflikt; vor Commit):

```
IF Z(Ti) < ZW(x)
   THEN NEUSTART
   ELSE ZR(x) := max ( ZR(x), Z(Ti) )   (* und Zugriff *)
```

- T_i will x schreiben (Test auf Schreib-Lese- oder Schreib-Schreib-Konflikt; nur während der Commit-Verarbeitung):

```
IF Z(Ti) < ZR(x)                    (* Schreib-Lese-Konflikt *)
OR Z(Ti) < ZW(x)                 (* Schreib-Schreib-Konflikt *)
```



```
THEN NEUSTART
ELSE ZW(x) := Z(T_i)                          (* und Zugriff *)
```

Die Zeitstempel an den Objekten ermöglichen einen sehr effizienten Validationstest, der bei dem ursprünglichen Korrektheitsbegriff Serialisierbarkeit nicht möglich wäre.

Das obige Kommando `NEUSTART` ist so zu interpretieren, daß T_i zurückgesetzt und anschließend automatisch neugestartet wird. Dabei erhält T_i einen *neuen* Zeitstempel.

Wir nehmen hier an, daß alle Validationstests automatisch innerhalb der Aktionen durchgeführt werden und nicht etwa von Hand durch den Programmierer. Alle Validationstests müssen unter wechselseitigem Ausschluß auf den betroffenen Daten durchgeführt werden.

2.2 Zyklischer Neustart

Eine Transaktion, die zurückgesetzt und mit neuem Zeitstempel neugestartet wurde, fängt völlig gleichberechtigt mit allen anderen noch vorhandenen Transaktionen von vorne an. Sie kann daher wieder Opfer eines Validationstests werden. In der Grundform des Zeitstempel-Verfahrens gibt es keine Möglichkeit, die Zahl der Neustarts zu begrenzen. Im schlimmsten Fall wird eine Transaktion immer wieder wegen neu hinzukommender Transaktionen abgebrochen[3]. Vor allem lange Transaktionen, die viele Lesezugriffe durchführen, können leicht das Opfer von kurzen schreibenden Transaktionen werden.

Wenn man die Wahrscheinlichkeit eines einzelnen Neustarts einer Transaktion als fest annimmt, dann werden häufige Neustarts zwar immer unwahrscheinlicher. Bei langen Transaktionen ist jedoch die Wahrscheinlichkeit eines einzelnen Neustarts relativ hoch. In jedem Fall muß mit unvertretbaren Ausreißern bei der Zahl der Neustarts gerechnet werden.

Unter gewissen Umständen können zwei Transaktionen immer wieder gegenseitig ihren Neustart verursachen. Beispiel:

[3] Ein ähnliches Problem besteht bei der Deadlock-Auflösung in Sperrverfahren.

```
T1 Z-r(x)----w(y)N
T2        Z-r(y)-------w(x)N
T1_1                   Z-r(x)-------w(y)N
T2_1                                Z-r(y)-------w(x)N
T1_2                                                 Z-r(x)--...
```

In diesem Beispiel wurde ein Zugriff, dessen Validationstest negativ ausging und zu einem Neustart führte, durch "N" dargestellt. Die neu gestarteten Transaktionen erhielten den Index 1, 2 usw. Das Beispiel läßt sich endlos fortführen, hier liegt ein zyklischer Neustart vor, bei dem nicht ständig neue Transaktionen hinzukommen müssen.

Wiederholter oder zyklischer Neustart sind ein Problem aller validierenden Verfahren. Bei Sperrverfahren existierte dieses Problem praktisch nicht. Daher wurden Sperrverfahren und validierende Verfahren kombiniert, um sowohl Deadlock-Freiheit wie Begrenzung der Neustarts zu erreichen. Einige Ansätze werden wir später behandeln.

2.3 Zeitstempel

Zeitstempel können einer Transaktion irgendwann zwischen ihrem Start durch ein Benutzerprogramm und ihrer ersten Aktion zugewiesen werden. Für unsere Betrachtungen ist die Zeit vor der Zuteilung des Zeitstempels irrelevant, d.h. für uns beginnt jede Transaktion mit der Zeitstempelzuteilung.

Zeitstempel waren oben zunächst eingeführt worden als Angabe einer Uhrzeit mit ausreichender Genauigkeit. Für Zeitstempel sind folgende Eigenschaften wesentlich:

1. Zwei Transaktionen haben nie den gleichen Zeitstempel.
2. Nichtüberlappende Transaktionen erhalten Zeitstempel in aufsteigender Reihenfolge.
3. Nachdem eine Transaktion ihren Zeitstempel erhalten hat, erhalten anschließend höchstens endlich viele andere Transaktionen einen kleineren ("älteren") Zeitstempel.

Die dritte Eigenschaft ist bei der Methode "Uhrzeit" in einer zentralen Datenbank trivialerweise erfüllt: Es gibt überhaupt keine Transakti-

on, die zukünftig einen kleineren Zeitstempel erhält. Die Abschwächung auf "endlich viele" ist wichtig für mehrere Prozessoren mit eigener Uhr, besonders also für verteilte Datenbanken, für die Zeitstempel-Verfahren ursprünglich entwickelt wurden und in denen u.U. nicht alle lokalen Uhren ausreichend synchronisiert werden können.

In zentralen Datenbanken bietet sich eine andere Lösung als die Uhrzeit an: Transaktionsnummern. Hierzu ist ein zentraler Transaktionszähler zu installieren, der bei jeder neuen Transaktion um eins erhöht wird. Transaktionsnummern haben gegenüber Uhrzeitangaben den Vorteil, daß zu ihrer Speicherung weniger Platz gebraucht wird.

2.4 Verwaltung der Zeitstempel

Gleichgültig, ob Uhrzeiten oder Transaktionsnummern als Zeitstempel verwandt werden, ist es nicht sinnvoll, bei jedem Objekt zwei Hilfsvariablen passender Größe zur Speicherung der Zeitstempel `ZR` und `ZW` fest einzurichten: Der Platzbedarf wäre sehr hoch, während andererseits nur ein Bruchteil dieser Zeitstempel wirklich benötigt wird. Benötigt werden nur noch solche Zeitstempel, die größer sind als der Zeitstempel der derzeit ältesten Transaktion, denn nur solche Zeitstempel können noch zum Rücksetzen einer Transaktion führen.

Betrachten wir zunächst die Lese-Zeitstempel getrennt. Die Lösung besteht in einer separaten Tabelle, die Einträge der Form (x, `ZR(x)`) enthält, allerdings nur für diejenigen Objekte x, deren `ZR` -Zeitstempel noch benötigt wird. Nicht mehr benötigte Einträge der Tabelle werden jeweils gelöscht. Zu der Tabelle gehört eine weitere Variable R_{min}, deren Wert kleiner als alle noch vorhandenen Zeitstempel in der Tabelle und größer als alle schon gelöschten Zeitstempel ist. Abfragen bzw. Änderungen von `ZR(x)` werden mit Hilfe der Tabelle wie folgt durchgeführt:

- Abfrage von `ZR(x)` : Die Tabelle wird nach einem Eintrag (x,z) durchsucht. Wird ein solcher Eintrag gefunden, so wird z zurückgegeben, andernfalls R_{min}.
- Änderung von `ZR(x)` : Die Tabelle wird nach einem Eintrag (x,z)

durchsucht. Wird er gefunden, so wird z durch den neuen Wert ersetzt, andernfalls wird ein neuer Eintrag eingefügt.

Bei bestimmten Gelegenheiten, z.B. bei Überlauf der Tabelle, wird ein neuer Wert für R_{min} festgelegt. Dieser Wert muß auf jeden Fall kleiner als der kleinste Zeitstempel aller noch aktiven Transaktionen sein. In der Tabelle werden dann alle Einträge (x,z) mit $z < R_{min}$ gelöscht.

Für die `ZW` -Zeitstempel muß eine weitere Tabelle mit einer zugehörigen Variablen W_{max} eingerichtet werden. Beide Tabellen können auch zusammengelegt werden. Diese Zeitstempel-Tabellen sind in einigen Details der Sperrtabelle sehr ähnlich.

3 Kombinierte Sperr- und Zeitstempel-Verfahren

Einige Verfahren benutzen sowohl Sperren wie auch Zeitstempelgesteuertes Rollback mit Neustart. Man kann sie als Sperrverfahren auffassen, in denen Zeitstempel-gesteuertes Rücksetzen zur Verhinderung von Deadlocks benutzt wird.

In [StLR76] und [RoSL78] wurden Verfahren vorgeschlagen, die das Deadlock- bzw. Neustart-Problem lösen, indem älteren Transaktionen höhere Priorität eingeräumt wird als jüngeren. Zunächst müssen Transaktionen das 2-Phasen-Protokoll befolgen. Jede Transaktion erhält bei ihrer Ankunft im System eine Zeitmarke. Sie behält diese Zeitmarke auch im Falle eines Neustarts (im Gegensatz zur Grundform der Zeitstempelverfahren)!

Die von den Zeitstempelverfahren her bekannten Lese- und Schreib-Zeitstempel an Objekten (`ZW(x)` und `ZR(x)`) werden nicht mehr direkt in dieser Form benutzt, jedoch indirekt durch Zeitstempel von Transaktionen, die in der Sperrtabelle verwaltet werden müssen.

Abweichend vom 2-Phasen-Protokoll gelten in dem Falle, daß T_1 eine Sperre beantragt, die nicht verträglich ist mit einer schon vorhandenen Sperre auf diesem Objekt, welche von T_2 gehalten wird, folgende Regeln:

wait-die-Verfahren:

- Wenn T_1 älter als T_2 ist, dann wartet T_1 auf die Freigabe der vorhandenen Sperre ("*wait*").
- Wenn T_1 jünger als T_2 ist, dann wird T_1 neu gestartet ("*die*").

wound-wait-Verfahren:

- Wenn T_1 älter als T_2 ist, dann wird T_2 neu gestartet; T_2 wird sozusagen von T_1 tödlich verwundet ("*wound*").
- Wenn T_1 jünger als T_2 ist, dann wartet T_1 auf die Freigabe der vorhandenen Sperre ("*wait*").

Das erste Stichwort in wait-die und wound-wait deutet an, was passiert, wenn eine ältere Transaktion auf eine jüngere warten soll, das zweite Stichwort den anderen Fall.

Deadlockfreiheit. Der Hauptunterschied zwischen beiden Verfahren ist die zulässige Wartebeziehung zwischen jüngeren und älteren Transaktionen:

- Beim wait-die-Verfahren wartet immer eine ältere Transaktion auf eine jüngere, nie umgekehrt.
- Beim wound-wait-Verfahren wartet immer eine jüngere Transaktion auf eine ältere, nie umgekehrt.

In beiden Fällen geben die Zeitstempel der Transaktionen eine lineare Ordnung vor (aufsteigende oder fallende Reihenfolge der Zeitstempel), in der sich die Wartebeziehungen aller Transaktionen bewegen können. Ein Wartezyklus ist somit unmöglich, die Verfahren sind deadlockfrei.

Neustarts. Gemeinsam ist beiden Verfahren, daß immer die *jüngere* Transaktion zurückgesetzt wird, wenn eine unzulässige Wartebeziehung droht, nie die ältere. Zyklische Neustarts sind damit ebenfalls unmöglich. Wesentlich hierbei ist, daß – im Gegensatz zur Grundform der Zeitstempel-Verfahren – eine Transaktion beim Neustart ihren ursprünglichen Zeitstempel behält. Sie kann also höchstens durch die

endlich vielen älteren Transaktionen zurückgesetzt werden. Diese wiederum werden in endlicher Zeit beendet, da kein Deadlock eintreten kann. (Auf eine spezielle Möglichkeit zur Endlosblockierung und ihre Verhinderung gehen wir anschließend ein.)

Beim wait-die-Verfahren ist die jüngere Transaktion die aktive Instanz, die einen Neustart auslöst, nämlich ihren eigenen. (Beim wound-wait-Verfahren ist dies die ältere Transaktion.) Hieraus resultiert ein Vorteil des wait-die-Verfahrens: Sobald eine Transaktion keine Sperren mehr anfordert, also alle benötigten Objekte besitzt, wird sie nicht mehr neugestartet. Beim wound-wait-Verfahren kann eine Transaktion dagegen jederzeit zurückgesetzt werden. Aus diesem Grund dürfte auch die Häufigkeit von Neustarts beim wound-wait-Verfahren höher sein.

Beim wait-die-Verfahren kann eine Transaktion, die zurückgesetzt wurde, wenn sie rasch neugestartet wird, in den gleichen Konflikt wie vorher mit einer älteren Transaktion kommen, die nach wie vor die gleichen Sperren hält. Besonders bei stark frequentierten, exklusiv zu sperrenden Objekten droht, daß sehr viele Transaktionen immer wieder wegen dieser Objekte scheitern. Hieraus leitet sich die Empfehlung ab, nicht zu schnell neu zu starten. Beim wound-wait-Verfahren besteht dieses Problem nicht.

Sperrenzuteilung. Ein weiteres Problem beim wait-die-Verfahren sind Endlosblockierungen[4]. So kann eine ältere Transaktion, die ein Objekt schreibsperren möchte und auf die Freigabe von vorhandenen Lesesperren wartet, von neu gewährten Lesesperren für dieses Objekt endlos blockiert werden. Für die Aspekte, die bei der Wahl einer Sperrenzuteilungsstrategie zu beachten sind, sei auf [SPV] verwiesen.

Bei beiden Verfahren muß in dem Fall, daß mehrere Transaktionen auf Freigabe der gleichen Sperre warten, die "Warterichtung" berücksichtigt werden, sonst können Deadlocks auftreten. Hierzu betrachten wir als Beispiel folgende Wartesituation nach dem Präfix eines Ablaufs beim wait-die-Verfahren (XLOCK(..) fordert Schreibsperren für die angegebenen Objekte an, ... steht für eine anschließende Wartezeit):

[4] vgl. Abschnitt 4.1 in [SPV].

```
T1 Z--------XLOCK(x,y).............
T2   Z-----r(y)-------XLOCK(x)...
T3     Z--r(x)--------------w(z)
```

T_1 und T_2 warten auf die Freigabe der Lesesperre auf x, die T_3 hält. T_1 wartet außerdem auf die Freigabe der Lesesperre auf y, die T_2 hält. Angenommen, T_3 endet und gibt die Sperre auf x frei. Nun kann entweder T_1 oder T_2 eine Schreibsperre für x gewährt werden. Im ersten Fall tritt ein Deadlock ein, die Sperre muß T_2, der jüngeren Transaktion gewährt werden.

Sofern bei der Zuteilung einer Sperre zwischen zwei Transaktionen zu entscheiden ist, muß bei beiden Verfahren die Transaktion bevorzugt werden, auf die die andere potentiell wartet, also beim wait-die-Verfahren die jüngere ("älter" wartet auf "jünger") und beim wound-wait-Verfahren die ältere.

Abschließend sei noch bemerkt, daß die Zeitstempel bei beiden Verfahren – im Gegensatz zur Grundform der Zeitstempel-Verfahren – keine Serialisierungspunkte sind.

Literatur

[RoSL78] Rosenkrantz, D.J.; Stearns, R.; Lewis, P.M.: System level concurrency control for distributed database systems; ACM ToDS 3:2, p.178-198; 1978/06

[StLR76] Stearns, R.E.; Lewis, P.M.; Rosenkrantz, L.J.: Concurrency control for database systems; p.19-32 in: Annual Symposium on Foundations of Computer Science; 1976

[REC] Kelter, U.: Lehrmodul "Recovery"; 2003

[SPV] Kelter, U.: Lehrmodul "Sperrverfahren"; 2003

Index