Optimistische Synchronisationskonzepte in zentralisierten und verteilten Datenbanksystemen

Concepts for optimistic concurrency control in centralized and distributed database systems

Erhard Rahm, Universität Kaiserslautern

Dipl.-Inform. **E. Rahm** studierte Informatik an der Universität Kaiserslautern. Seit Ende 1984 ist er dort wissenschaftlicher Mitarbeiter am Fachbereich Informatik und steht zur Zeit vor seiner Promotion. Interessen- und Forschungsschwerpunkte: Mehrrechner-Datenbanksysteme, Leistungsbewertung von Datenbanksystemen sowie Synchronisation, Lastkontrolle und Recovery.

Anschrift des Verfassers: Universität Kaiserslautern, FB Informatik, Postfach 3049, D-6750 Kaiserslautern.

Optimistische Synchronisationsverfahren werden – vor allem für konfliktärmere Datenbankanwendungen – zunehmend als eine vielversprechende Alternative zu Sperrverfahren angesehen. In dieser Arbeit werden für zentralisierte und verteilte Datenbanksysteme sowohl die wichtigsten der bereits veröffentlichten Einsatzformen einer optimistischen Synchronisation überblicksartig angegeben als auch eine Reihe neuer und verbesserter Verfahren und Techniken vorgestellt. Nach der Einführung werden neben dem für zentralisierte Datenbanksysteme vorgeschlagenen ursprünglichen BOCC-Verfahren zwei stark verbesserte Alternativen (der FOCC- und der BOCC+-Ansatz) sowie eine Kombinationsmöglichkeit mit Sperrverfahren untersucht. Danach wird gezeigt, wie diese grundlegenden Algorithmen zur Synchronisation in verteilten Datenbanksystemen einsetzbar sind. Dabei können die Validierungen entweder an zentraler Stelle oder verteilt erfolgen, wobei vor allem letztere Organisationsform neue Probleme aufwirft. In einem weiteren Kapitel werden dann noch drei wichtige Optimierungsmöglichkeiten für optimistische Synchronisationsprotokolle angesprochen: das Vorverlegen von Lese-Transaktionen in der Serialisierungsreihenfolge, Einsatz eines Multiversion-Konzepts und Beschränkung auf Konsistenzebene 2.

Optimistic synchronization methods are considered as a promising alternative to locking algorithms for concurrency control, especially for applications with moderate conflict probability. This paper aims at providing a survey of some existing proposals for optimistic synchronization in centralized and distributed database systems, as well as describing a number of new and improved techniques. After an introduction we review the original BOCC-scheme for centralized database systems and describe two heavily improved alternatives (the FOCC and the BOCC+ approach) and also a combination with locking. We then show how these basic algorithms can be used for synchronization in distributed database systems. Here the validations can be performed either at a central site or in a distributed way where the latter approach introduces some extra problems to solve. Furthermore, three important optimizations for optimistic synchronization protocols are discussed: placing transactions before the end of the serialization order in case of conflict, use of a multiversion concept and restriction to level-2-consistency.

1. Einführung

Bei der Realisierung eines Datenbanksystems (DBS), und insbesondere eines verteilten DBS, spielt die Synchronisationskomponente eine zentrale Rolle. Sie hat die Serialisierbarkeit der parallel bearbeiteten Transaktionen (TA) zu gewährleisten, damit Anomalien und Datenbankinkonsistenzen im Mehrbenutzerbetrieb verhindert werden. Serialisierbarkeit (logischer Einbenutzerbetrieb) liegt dabei dann vor, wenn zu der parallelen Bearbeitung der TA eine äquivalente serielle Ausführungsreihenfolge (oder Schedule) existiert, die für jede der beteiligten TA die gleiche Ausgabe sowie den gleichen Datenbankendzustand erzeugt [1]. Um hohe TA-Raten und kurze Antwortzeiten zu ermöglichen, sollte die Synchronisation mit einem Mindestmaß an Blockierungen und Rücksetzungen von TA durchgeführt werden. In verteilten DBS (VDBS) ist aus Leistungsgründen außerdem der durch das Synchronisationsprotokoll eingeführte Kommunikationsaufwand weitgehend ein-

zugrenzen. Vor allem ist die Anzahl „synchroner" Nachrichten, für die eine TA auf eine Antwortnachricht warten muß, minimal zu halten, weil solche Nachrichten unmittelbar die Antwortzeiten verschlechtern und zusätzlichen Aufwand zum Deaktivieren und späteren Aktivieren der TA einführen (Prozeßwechsel).

In dieser Arbeit werden mögliche Einsatzformen sogenannter optimistischer Synchronisationsverfahren [2] untersucht, die vor einigen Jahren als Alternative zu den üblicherweise benutzten Sperrverfahren vorgeschlagen wurden. Sie sollen u.a. die folgenden Schwächen, die Sperrverfahren nachgesagt werden, beseitigen:

- Sperrverfahren sind zu pessimistisch, da sie referenzierte Objekte immer sperren, obwohl Sperren nur im Konfliktfall benötigt werden. Der damit verbundene Verwaltungsaufwand ist nur gerechtfertigt, wenn viele Konflikte zu erwarten sind.
- Die Erkennung von Deadlocks verursacht einen hohen Overhead, vor allem in VDBS.
- Um Serialisierbarkeit garantieren zu können, müssen Sperren i.a. bis Transaktionsende (EOT) gehalten werden. Daher nehmen die Anzahl der Konflikte sowie die Dauer von Blockierungen mit der Transaktionslänge zu.
- Sperrkonflikte führen nicht nur zu Antwortzeitverschlechterungen, sondern auch zu Durchsatzeinbußen (Herabsetzung der Parallelität durch blockierte TA).
- In Mehrrechner-DBS führen einige Sperrverfahren zu vielen externen Sperranforderungen (hoher Kommunikationsaufwand).

Optimistische Synchronisationsverfahren gehen von der expliziten Annahme aus, daß Konflikte zwischen TA seltene Ereignisse darstellen und somit der mit Sperrverfahren verbundene Aufwand nicht notwendig ist. Daher greifen optimistische Verfahren zunächst nicht in den Ablauf von TA ein, sondern erlauben ein nahezu beliebig paralleles Arbeiten auf der Datenbank. Erst bei Transaktionsende wird überprüft, ob Konflikte mit anderen TA aufgetreten sind.

Gemäß dieser Vorgehensweise unterteilt man die Ausführung einer TA in drei Phasen: In der **Lesephase** wird die eigentliche TA-Verarbeitung vorgenommen, d.h., es werden Objekte der Datenbank gelesen und modifiziert. Jede TA führt dabei alle ihre Änderungen in einem ihr zugeordneten TA-Puffer aus, der für keine andere TA zugänglich ist.

Bei EOT wird eine **Validierungsphase** gestartet, in der geprüft wird, ob die beendigungswillige TA mit einer parallel zu ihr laufenden TA in Konflikt geraten ist. Ein Konflikt wird stets durch Zurücksetzen einer oder mehrerer beteiligter TA aufgelöst.

Die **Schreibphase** wird nur von Änderungstransaktionen ausgeführt, die die Validierungsphase erfolgreich beenden konnten. In dieser Phase muß – z.B. durch Logging – sichergestellt werden, daß die Änderungen der TA nicht mehr verlorengehen können (etwa durch einen Rechner- oder Plattenausfall). Außerdem sind alle Änderungen für andere TA sichtbar zu machen (update propagation).

Die bisher vorgenommene Beschreibung optimistischer Synchronisationsverfahren läßt offensichtlich noch einen großen Spielraum für mögliche Realisierungsformen (insbesondere bei der Validierung), deren Beschreibung einen wesentlichen Inhalt dieses Aufsatzes darstellt. Dennoch lassen sich bereits jetzt eine Reihe von Eigenschaften optimistischer Verfahren festhalten:

- TA-Rücksetzungen sind sehr einfach; es brauchen nur die Inhalte der TA-Puffer gelöscht zu werden (keine E/A).
- Es können keine Deadlocks entstehen.
- Die TA-Puffer bedingen einen zusätzlichen Speicheraufwand.
- Konflikte werden nicht durch Blockierung von Transaktionen, sondern ausschließlich durch Zurücksetzen aufgelöst. Es ist daher mit mehr Rücksetzungen als bei Sperrverfahren zu rechnen.

Obwohl optimistische Verfahren noch in keinem kommerziellen DBS verwendet werden, läßt sich das starke Interesse an ihnen an einer Vielzahl von Veröffentlichungen ablesen. Darüber hinaus werden sie bereits in einer Reihe von Prototypen eingesetzt, vor allem in verteilten Umgebungen [3–7].

Im nächsten Abschnitt werden zunächst der ursprüngliche Vorschlag aus [2] sowie einige verbesserte optimistische Verfahren für zentralisierte DBS vorgestellt. Danach werden dann für VDBS in Kapitel 3 und 4 optimistische Synchronisationsprotokolle mit zentraler und verteilter Validierung angegeben. Nach einer Darstellung verschiedener Optimierungsmöglichkeiten in Kapitel 5 werden dann die wichtigsten Erkenntnisse noch einmal zusammengefaßt.

Der Schwerpunkt dieser Arbeit liegt in der Vermittlung der bei optimistischer Synchronisation anwendbaren Konzepte sowie deren qualitativen Beurteilung. Quantitative Bewertungen, v.a. für die vorgeschlagenen Verbesserungen, können hier jedoch nicht vorgenommen werden, da hierzu i.a. nur detaillierte und realitätsnahe Simulationen weiterhelfen.

Zu den in dieser Arbeit erstmals beschriebenen Verfahren und Techniken zählen unter anderem eine einfache und effektive Realisierungsmöglichkeit für parallele Schreibphasen (Kap. 2.2), das BOCC+-Verfahren (2.3), ein zentrales Validierungsschema bei VDBS (Kap. 3) sowie die Verwendung von optimistischen Protokollen zur Gewährleistung von Konsistenzebene 2 (in 5.3). Die Lösungsmöglichkeiten bei verteilter Validierung in Kapitel 4 werden an einem einheitlichen Verarbeitungsmodell und zum Teil in erweiterter Form dargestellt.

2. Optimistische Synchronisation in zentralisierten Datenbanksystemen

Zunächst sollen hier einige allgemeine Annahmen und Vereinbarungen beschrieben werden, die dann bei der Beschreibung der Algorithmen Verwendung finden.

Um eine Validierung durchführen zu können, werden für jede Transaktion T_i während ihrer Lesephase die Namen von ihr gelesener bzw. veränderter Objekte in einem Read-Set (RS_i) bzw. Write-Set (WS_i) geführt. Es wird angenommen, daß jedes Objekt vor einer Änderung gelesen wird ($WS \subseteq RS$), d.h., es gibt keine „blinden Schreiber". Bei optimistischen Synchronisationsverfahren wird zumeist verlangt, daß die Serialisierungsreihenfolge der TA der Validierungsreihenfolge entspricht. Das bedeutet, daß eine validierende TA im äquivalenten seriellen Schedule nicht mehr vor bereits validierten TA eingefügt werden kann, sondern stets an das Ende. Die Reihenfolge einer TA innerhalb des seriellen Schedules kann daher bei diesen Verfahren in einfacher Weise von einem Zähler TNC (transaction number counter) abgeleitet werden, der nach jeder erfolgreichen Validierung um 1 inkrementiert wird. Die validierende TA T_i bekommt dabei den aktuellen Wert von TNC als ihre TA-Nummer $n(T_i)$ zugewiesen, die zugleich ihre Position im äquivalenten seriellen Schedule festlegt. So bedeutet $n(T_j) < n(T_i)$, daß T_j vor T_i validiert hat und T_j daher auch im seriellen Schedule vor T_i kommt ($T_j \rightarrow T_i$). Zur korrekten Synchronisation ist im wesentlichen zu gewährleisten, daß keine TA sich erfolgreich beenden kann, wenn sie Daten gelesen hat, die danach von zwischenzeitlich beendeten TA geändert in die Datenbank eingebracht wurden.

Nach [8] lassen sich optimistische Synchronisationsverfahren gemäß ihrer Validierungsstrategie grob in zwei Klassen unterteilen. Bei den rückwärtsorientierten Verfahren (Backward Oriented Optimistic Concurrency Control, **BOCC**), die als erstes vorgeschlagen wurden [2], erfolgt die Validierung ausschließlich gegenüber bereits beendeten Transaktionen. Bei den vorwärtsorientierten Verfahren (Forward Oriented Optimistic Concurrency Control, **FOCC**) dagegen wird gegen noch laufende Transaktionen validiert. Zunächst wird nun in 2.1 auf den ursprünglichen BOCC-Vorschlag sowie die mit ihm verbundenen Nachteile eingegangen. Danach folgt die Beschreibung von besseren optimistischen Verfahren, nämlich von FOCC-Protokollen (2.2) sowie dem sogenannten BOCC+-Ansatz (2.3). Abschließend wird dann noch in 2.4 eine Kombination mit Sperrverfahren diskutiert.

2.1 Das ursprüngliche BOCC-Verfahren nach [2]

Bei diesem Verfahren muß jede TA sich der Validierung unterziehen, wobei überprüft wird, ob die validierende TA ein Objekt gelesen hat, das während ihrer Lesephase geändert wurde. Dazu wird in der Validierungsphase einer TA T_j überprüft, ob ihr Read-Set RS_j sich mit dem Write-Set irgendeiner TA überschneidet, die parallel zu T_j gelaufen und bereits erfolgreich beendet ist. Ist dies der Fall, so wird T_j zurückgesetzt, da möglicherweise auf veraltete Daten zugegriffen wurde (die am Konflikt beteiligten TA können nicht mehr zurückgesetzt werden, da sie bereits beendet sind).

Um den angesprochenen Test durchführen zu können, vermerkt sich jede TA T_j den Wert von TNC bei BOT in $TSTART_j$ und den Wert bei Beginn der Validierungsphase in $TFINISH_j$. Die Validierung für T_j, die zusammen mit der Schreibphase in einem kritischen Abschnitt stattfindet (durch ≪ ≫ gekennzeichnet), sieht dann aus wie folgt:

```
  VALID := true;
≪TFINISHj := TNC;
  for n(Ti) from TSTARTj + 1 to TFINISHj do;
      if RSj ∩ WSi ≠ ∅ then VALID := false;
  end;
  if VALID then do;
      TNC := TNC+1;
      n(Tj) := TNC;
      Schreibphase für Tj; ≫
  end;
  else (setze Tj zurück);
```

Dieser einfache Algorithmus besitzt allerdings eine Reihe schwerwiegender **Nachteile,** so daß er bestenfalls für äußerst konfliktarme Anwendungen in Betracht kommt (z.B. wenn nur selten Änderungstransaktionen gestartet werden):

- Der **kritische Abschnitt** bezieht sich zwar nur auf andere validierungsbereite TA (ein Lesen von DB-Objekten ist weiterhin möglich), doch dauert er wegen der Schreibphase sehr lang und führt somit zu Behinderungen und möglichen Durchsatzbeschränkungen. Denn selbst wenn das Sichtbarmachen der Änderungen vollständig im Systempuffer vorgenommen werden kann, ist in der Schreibphase mindestens ein physischer E/A-Vorgang für Logging (oder äquivalente Maßnahme) notwendig (dieser Punkt wird in der Literatur häufig ignoriert). In [2] wurde bereits eine Erweiterung des Verfahrens vorgestellt, mit dem parallele Validierungs- und Schreibphasen möglich sind, allerdings auf Kosten einer erhöhten Rücksetzgefahr.
- Bei dem Verfahren führen selbst **„unechte" Konflikte** zu Rücksetzungen. Dies ist dann der Fall, wenn die Änderung einer parallelen TA bereits vor dem ersten Lesen in die DB eingebracht wurde. Dann wird nämlich die TA (unnötigerweise) zu-

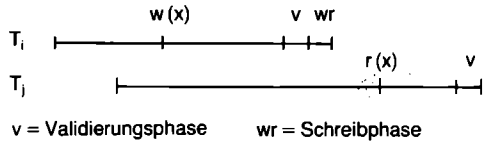

Bild 1. Rücksetzung von T_j wegen „unechtem" Konflikt.

rückgesetzt, obwohl die aktuelle Version gesehen wurde. So wird in dem Beispiel in Bild 1 T_j zurückgesetzt, obwohl sie die von T_i erzeugte Änderung von x mitbekommen hat.

Diese unnötigen Rücksetzungen treffen vor allem lange TA, die bereits durch kurze Änderungs-TA zum Scheitern gezwungen werden können. Denn das Verfahren berücksichtigt nicht die tatsächliche Zeitdauer, in der TA parallel zueinander in Bearbeitung waren; vielmehr wird der schlechteste Fall angenommen, daß die TA während ihrer gesamten Lebensdauer parallel abliefen [9].

- Selbst ohne Rücksetzungen aufgrund unechter Konflikte ist das BOCC-Verfahren denkbar **unfair** gegenüber langen TA. Denn für diese TA ist ein Validierungskonflikt sehr wahrscheinlich, da sie selbst einen relativ großen Read-Set haben und da sie sich wegen ihrer Dauer im allgemeinen gegenüber vielen TA validieren müssen. Weil bei einem Konflikt immer die validierende TA zurückgesetzt wird, ist es möglich, daß vor allem lange TA immer wieder scheitern (**zyklischer Restart** oder Starvation-Problem). Das gleiche Problem besteht auch für TA, die auf vielgeänderte Daten zugreifen **(Hot Spots).**
- Nachteilig ist weiterhin, daß der **Write-Set** einer beendeten TA T (im Prinzip) **aufzuheben** ist, und zwar so lange, bis keine TA mehr im System ist, die sich gegen T validieren muß. Außerdem erfordert der Algorithmus eine relativ **hohe Anzahl von Vergleichen** während der Validierungsphase (jedes Element des Read-Sets der validierenden TA ist zu vergleichen mit jedem Element der zu betrachtenden Write-Sets).
- Eine weitere Schwäche des BOCC-Algorithmus liegt darin, daß bereits **zum Scheitern verurteilte TA nicht sofort,** sondern erst nach ihrer Validierungsphase **zurückgesetzt** werden, wodurch unnötige Arbeit verrichtet wird (Durchsatzeinbußen). Denn nach der erfolgreichen Validierung einer Änderungs-TA T_i ist klar, daß alle laufenden TA, die bereits ein Objekt aus WS_i gelesen haben, zurückgesetzt werden müssen. Dennoch werden die Lesephasen dieser TA noch vollständig ausgeführt.

Wie analytische Leistungsuntersuchungen [10–13] sowie Simulationen [14–17] gezeigt haben, führen die genannten Nachteile zu deutlichen Leistungseinbußen im Vergleich zu Sperrverfahren. Lediglich in nahezu konfliktfreien Anwendungen konnte das BOCC-Verfahren in etwa gleichwertige Ergebnisse erreichen, während ansonsten eine sehr hohe Anzahl von Rücksetzungen sehr schlechte Durchsatz- und Antwortzeitresultate verursachte. Dennoch bedeutet dies nicht eine prinzipielle Unterlegenheit des optimistischen Ansatzes im Vergleich zu Sperrverfahren. Vielmehr lassen sich die meisten der genannten Nachteile relativ einfach durch verbesserte optimistische Verfahren beheben, wie im folgenden gezeigt wird. Eines der Hauptprobleme, die Vermeidung zyklischer Restarts, scheint jedoch im allgemeinen nur in Kombination mit Sperrverfahren behebbar (2.4). In Kapitel 5 werden darüber hinaus noch weitere Optimierungsmöglichkeiten zur Reduzierung der Anzahl von Rücksetzungen angesprochen, die sowohl im zentralisierten Fall als auch für VDBS anwendbar sind.

2.2 FOCC-Verfahren

Mit den FOCC-Verfahren, die in [18, 19] unter dem Namen Schnappschuß-Validation eingeführt wurden, können einige der BOCC-Nachteile umgangen werden. Bei ihnen wird nicht gegen bereits beendete TA validiert, sondern gegen sämtliche aktiven TA. In der Validierungsphase, die nur von Änderungs-TA durchzuführen ist, wird untersucht, ob irgendeine der in der Lesephase befindlichen TA ein Objekt gelesen hat, das die validierende TA im Begriff ist zu ändern. In diesem Fall muß der Konflikt durch Zurücksetzen einer (oder mehrerer) beteiligter TA aufgelöst werden.

Die EOT-Bearbeitung für eine Änderungs-TA T_j läßt sich demnach folgendermaßen darstellen:

```
VALID := true;
≪for (alle laufenden TA Ti, die noch nicht validiert) do;
    if WSj ∩ RSi ≠ ∅ then VALID := false;
end;
if VALID then Schreibphase für Tj; ≫
else (löse Konflikt auf);
```

Auf die Führung eines TA-Zählers TNC wird bei FOCC verzichtet, da er hier nicht benötigt wird, um die TA zu bestimmen, gegen die validiert werden muß.

Der FOCC-Ansatz bietet einige Vorteile gegenüber dem ursprünglichen BOCC-Verfahren:

- keine Rücksetzungen wegen „unechten" Konflikten
- kein Aufbewahren von Write-Sets beendeter TA notwendig
- weniger Vergleiche zur Validierung (etwa die Hälfte [17]), weil gegen noch laufende TA validiert wird, deren Read-Set im Mittel erst zur Hälfte feststeht
- Konflikte werden frühzeitig erkannt, wodurch unnötige Arbeit eingespart werden kann.

Der vielleicht größte Vorteil des FOCC-Verfahrens ist jedoch, daß es eine **größere Flexibilität** bei der Auflösung von Konflikten gestattet, da noch keine der am Konflikt beteiligten TA beendet ist.

Dies erlaubt mehr Fairneß bei der Auswahl der „Opfer" und eine Linderung des Starvation-Problems. Zur Auflösung eines Konfliktes lassen sich nach [8] vor allem folgende Alternativen unterscheiden:

1. **Kill:** Hierbei werden alle laufenden TA, deren Read-Sets sich mit dem Write-Set der validierenden TA überschneiden, zurückgesetzt.
2. **Abort:** Die validierende TA wird zurückgesetzt.
3. **Defer:** Die validierende TA wird so lange verzögert, bis alle in Konflikt stehenden TA beendet sind.
4. **Hybride Strategie** aus den vorgenannten Alternativen.

Wie man leicht einsieht, erscheint am ehesten eine hybride Strategie sinnvoll, da die drei erstgenannten Alternativen für sich allein zu einseitig sind und einen zyklischen Restart von TA nicht ausschließen können. So ist die Kill-Strategie vor allem gegenüber langen (Lese-)TA unfair, für die – ähnlich wie bei BOCC – eine hohe Rücksetzwahrscheinlichkeit besteht. Die Abort-Strategie garantiert Lese-TA ihr Durchkommen, benachteiligt jedoch Änderungs-TA (vor allem wenn sie lang sind bzw. einen „Hot Spot" ändern wollen). Die Defer-Methode garantiert ebenfalls Lesern immer ein erfolgreiches Ende, allerdings widerum auf Kosten der Änderungs-TA, die der Gefahr einer endlosen Verzögerung ausgesetzt sind. Ein grundsätzlicher Nachteil der Defer-Methode ist zudem, daß durch sie Deadlocks entstehen können. Mit einer hybriden Strategie (aus Kill- und Abort-Methode) lassen sich dagegen zyklische Restarts vermeiden, indem man TA unterschiedliche Prioritäten zuordnet, z. B. abhängig von Laufzeit oder Anzahl der Rücksetzungen. Somit kann immer mindestens einer TA (der mit der höchsten Priorität) ein gesichertes Durchkommen ermöglicht werden.

Ein Problem bei FOCC stellt allerdings der **kritische Abschnitt** dar, der sich hier nicht nur auf andere validierungsbereite TA bezieht, sondern während jeder Validierungs- und Schreibphase alle Zugriffe auf die Datenbank ausschließt. Dies wird deshalb vorgesehen, da sich ansonsten die Read-Sets der laufenden TA ändern und mögliche Konflikte unentdeckt bleiben könnten. Da die Praktikabilität des FOCC-Ansatzes mit einem solchen kritischen Abschnitt sicher nicht gegeben ist, wird im folgenden eine einfache, jedoch effektive Lösung für dieses Problem angegeben.

Die grundlegende Beobachtung ist die, daß der kritische Abschnitt praktisch einer Sperre für die gesamte Datenbank gleichkommt. Mehr Parallelität läßt sich daher erzielen, indem man nicht die gesamte Datenbank blockiert, sondern nur noch die Objekte, die eine Änderungstransaktion schreiben möchte (dies entspricht einem kurzzeitigen Setzen von X-Sperren auf die betroffenen Objekte). Durch das „Sperren" der Objekte wird automatisch verhindert, daß zwei TA ein Objekt zur gleichen Zeit schreiben wollen, so daß die Schreibphase ohne kritischen Abschnitt durchgeführt werden kann. Der Blockierungsvermerk für die betroffenen Objekte kann z. B. in einer Objekttabelle eingetragen werden, die vor jedem Zugriff auf die Datenbank (nicht jedoch für Zugriffe im TA-Puffer) zu inspizieren ist. Wird dabei ein Blockierungsvermerk entdeckt, reiht sich die TA in eine objektspezifische Warteliste ein. Eine Aktivierung der TA erfolgt dann, sobald die Änderung in die Datenbank eingebracht ist. Der genaue Aufbau von Validierungs- und Schreibphase für eine (Änderungs-)TA T_j sieht nun folgendermaßen aus:

```
≪ WSj blockieren;
  for (alle laufenden Ti, die noch nicht validiert)
    do;
    if WSj ∩ RSi ≠ ∅ then (löse Konflikt auf);
  end; ≫
  if (Tj erfolgreich) then do;
     Schreibphase für Tj;
     Blockierung für WSj aufheben; (* ggf. wartende TA aktivieren *)
  end;
  else do;
       Blockierung für WSj aufheben
       setze Tj zurück;
  end;
```

Wie zu sehen ist, wird die Blockierung des Write-Sets bereits vor der Validierung durchgeführt, um auch während der Validierungsphase ein Weiterverarbeiten auf den nicht blockierten Objekten zu ermöglichen. (Die Blockierung des Write-Set wurde im kritischen Abschnitt vorgenommen, um Deadlocks zu vermeiden. Diese könnten auch verhindert werden, wenn die Objekte in einer festgelegten Reihenfolge blockiert werden.) Der angegebene kritische Abschnitt bezieht sich daher nicht auf in der Lesephase befindliche TA, sondern lediglich (wie bei BOCC bzw. BOCC+) auf andere validierungsbereite TA. Dadurch, daß immer nur eine Validierung zu einem Zeitpunkt stattfindet, wird gewährleistet, daß die validierende TA alle noch nicht validierten TA berücksichtigen kann.

Trotz der genannten Vorteile des FOCC-Ansatzes besteht jedoch auch hier die Gefahr einer hohen Rücksetzrate, insbesondere wenn Hot Spots vorliegen. Daher sind auch für FOCC eine Kombination mit Sperrverfahren sowie weitere Optimierungsmöglichkeiten (wie etwa in Kap. 5 genannt) in die Überlegungen einzubeziehen.

2.3 Das BOCC+-Verfahren

BOCC+ steht für eine andere Verbesserung des ursprünglichen BOCC-Verfahrens, bei dem (wie der Name schon andeutet) wiederum eine rückwärtsorientierte Validierung gegenüber schon beendeten TA vorgenommen wird. Zugriffskonflikte werden hierbei jedoch über Zeitmarken

(ähnlich wie z.B. in [20, 21] vorgeschlagen) entdeckt. Dazu wird bei Änderung eines Objektes x die TA-Nummer der (erfolgreich) ändernden TA als Zeitstempel TS(x) bei dem Objekt selbst gespeichert. Beim Zugriff auf ein Objekt vermerkt sich nun jede TA zusätzlich diesen Zeitstempel zusammen mit ihrem Read-Set. Bei der Validierung einer TA T_j braucht dann nur durch Vergleich der Zeitstempel überprüft zu werden, ob die TA stets die aktuellste Version aller referenzierten Objekte gesehen hat oder nicht. Damit ist zugleich sichergestellt, daß nur echte Konflikte zur Rücksetzung einer TA führen. Die Validierungs- und Schreibphase für T_j sehen daher aus wie folgt ($ts_j(x)$ bezeichne den Zeitstempel von x beim Zugriff von TA T_j):

```
VALID := true;
≪for all r ∈ RSj do:
    if tsj(r) < TS(r) then VALID := false;
end;
if VALID then do;
        TNC := TNC+1;
        n (Tj) := TNC
        for all w ∈ WSj do;
          TS(w) := n(Tj);
        end;
        Schreibphase für Tj; ≫
end;
else (setze Tj zurück);
```

Der Algorithmus vermeidet nicht nur Rücksetzungen aufgrund unechter Konflikte, sondern erlaubt auch eine erhebliche Beschleunigung der Validierungsphase. Denn für jede TA braucht für jedes Element ihres Read-Set nur noch ein Vergleich vorgenommen zu werden, unabhängig von der Anzahl parallel ablaufender Änderungs-TA. Wie leicht einzusehen ist, ergeben sich so auch weitaus weniger Vergleiche als bei FOCC. Die Verwendung der Zeitstempel erlaubt es auch, die Schreibphasen außerhalb des kritischen Abschnittes durchzuführen. Denn falls eine TA in der Lesephase auf die alte Version eines zu schreibenden Objektes zugreift, wird dies durch Zeitstempelvergleich in der Validierungsphase festgestellt. Parallele Schreibphasen sind somit auch nur zwischen TA möglich, die unterschiedliche Objektmengen schreiben.

Der angegebene Algorithmus ist jedoch nur dann einsetzbar, wenn für jedes bei der Validierung zu berücksichtigende Objekt x der Zeitstempel TS(x) im Hauptspeicher zugreifbar ist. Da der Systempuffer nur begrenzte Kapazität besitzt, kann vor allem bei langen TA in der Regel nicht davon ausgegangen werden, daß alle während ihrer Laufzeit geänderten Objekte noch im Systempuffer vorliegen (ein Einlesen der Objekte zur Validierung verbietet sich von selbst). Daher werden in einer sogenannten **Objekttabelle,** die z.B. als Hash-Struktur organisiert sein kann, für zuletzt geänderte Objekte die Zeitstempel vermerkt. Ein Objekteintrag einer solchen Tabelle kann dabei gelöscht werden, sobald sichergestellt ist, daß keine TA mehr zu validieren ist, die zum letzten Änderungszeitpunkt des Objektes aktiv war. Demnach ist für Objekte, für die bei der Validierung kein Eintrag in der Objekttabelle gefunden wird, kein Konflikt möglich.

Das BOCC+-Verfahren läßt sich durch die zwei folgenden Maßnahmen noch weiter verbessern:

a) Ähnlich wie bei einem FOCC-Verfahren mit Kill-Strategie kann eine erfolgreich validierte Änderungs-TA T_j sofort alle laufenden TA zurücksetzen, die auf ein Objekt aus WS_j zugegriffen haben. Denn da diese TA zum Scheitern verurteilt sind, kann durch ihr frühzeitiges Zurücksetzen unnötige Arbeit eingespart werden.
b) Auch hier ist es (wie bei FOCC) sinnvoll, während einer Schreibphase zu ändernde Objekte für TA in der Lesephase zu blockieren. Denn dann werden Zugriffe auf die noch ungeänderten Objektversionen und somit vermeidbare Rücksetzungen verhindert.

Insgesamt besitzt der BOCC+-Ansatz, verglichen mit dem ursprünglichen BOCC-Verfahren, vor allem folgende Pluspunkte:

- keine Rücksetzungen aufgrund unechter Konflikte
- sehr schnelle Validierung (wenig Vergleiche)
- parallele Schreibphasen
- frühzeitige Zurücksetzung zum Scheitern verurteilter TA.

Verglichen mit FOCC schlägt die besonders schnelle Validierung positiv zu Buche sowie – wie sich noch zeigen wird – eine weitergehende Übertragbarkeit auf VDBS. Von Nachteil ist die fehlende Wahlmöglichkeit bei der Auflösung von Konflikten, so daß die Gefahr zyklischer Restarts gegeben ist. Dieses Problem läßt sich durch eine Kombination mit Sperrverfahren, die im nächsten Abschnitt diskutiert wird, lösen. Die in Kap. 5 vorgestellten Optimierungsmöglichkeiten lassen sich auch für BOCC+ anwenden.

2.4 Kombination mit Sperrverfahren

Eine Kombination von optimistischen Verfahren mit Sperrverfahren ist in erster Linie zur Vermeidung zyklischer Restarts (Starvation-Problem) sowie zur Reduzierung von Rücksetzungen wichtig. Diese Probleme sind vor allem relevant bei langen TA sowie im Falle von Änderungen unterworfenen Hot Spots (hohe Konfliktwahrscheinlichkeit). Während FOCC-Verfahren immer mindestens einer TA ein Durchkommen garantieren können und somit ein zyklischer Restart vermeidbar ist, bedarf der BOCC-Ansatz in jedem Fall einer expliziten Lösung des Starvation-Problems. Denn da bei BOCC und BOCC+ im Konfliktfalle stets die validierende TA zurückgesetzt werden

muß, sind ohne besondere Vorkehrungen zyklische Restarts durchaus möglich. Weil die das Starvation-Problem verursachende Konflikthäufigkeit offenbar nicht in Einklang mit der optimistischen Grundannahme konfliktarmer Verarbeitungsfolgen steht, sind zu dessen Lösung pessimistischere Strategien angebracht, d.h., es sind auch Blockierungen von TA vorzunehmen statt ausschließlich Rücksetzungen. Ein erster Vorschlag in diese Richtung war bereits das Blockieren von Objekten während der Schreibphase, wodurch sich vermeidbare Rücksetzungen umgehen lassen.

Es ist klar, daß die Unterstützung beider Verfahrensklassen zur Synchronisation einen hohen Realisierungsaufwand impliziert (z.B. ist eine Deadlock-Behandlung vorzusehen); andererseits können aber auch im Prinzip die Vorteile beider Strategien je nach Lastprofil genutzt werden. So ist bei einer höheren Konfliktwahrscheinlichkeit (z.B. falls Hot Spots erkennbar werden) eine pessimistischere Synchronisation zur Reduzierung von TA-Rücksetzungen angebracht, während man sich den damit verbundenen Overhead in konfliktarmen Situationen durch eine optimistischere Synchronisation ersparen kann. Die kombinierten Verfahren erlangen somit potentiell die gleiche Universalität wie Sperrverfahren mit einer möglicherweise besseren Leistungsfähigkeit für bestimmte Anwendungen oder Lastprofile. Dies wird durch erste Simulationsergebnisse in [22] bestätigt, bei denen ein kombiniertes Verfahren, wobei Lese-TA optimistisch und Änderungs-TA pessimistisch synchronisiert werden, stets besser abschnitt als das Zweiphasen-Sperrprotokoll.

Bei den bisher bekanntgewordenen Vorschlägen zur Integration von Sperrverfahren und optimistischen Methoden [23–25] erfolgt die Wahl des Synchronisationsverfahrens entweder auf der Ebene von Transaktionen bzw. auf der Ebene der Objekte. So kann beispielsweise mit ersterer Methode einer TA ihr Durchkommen (bei pessimistischer Synchronisation) zugesichert werden (sinnvoll vor allem für lange bzw. bereits gescheiterte TA), während der zweite Ansatz vor allem bei Hot Spots sinnvoll ist. Eine weitergehende Unterscheidung [24, 25], auf die hier jedoch nicht eingegangen werden kann, berücksichtigt zusätzlich noch die Art der Konflikte (Lese-Schreib- oder Schreib-Schreib-Konflikt). Eine spezielle Kombination von optimistischer und pessimistischer Synchronisation wird darüber hinaus bei IMS Fast Path für eine bestimmte Klasse von Hot-Spot-Objekten (den sogenannten High-Traffic-Elementen) benutzt [26, 27].

In dieser Arbeit soll nur kurz auf die einfachste Kombination mit Sperrverfahren – auf der Ebene von TA – eingegangen werden. Wir gehen dabei von den zwei üblichen Sperrtypen (Lese- und Schreibsperren) sowie von langen Sperren (Freigabe bei EOT) aus. Die Änderungen pessimistischer TA werden ebenfalls in privaten TA-Puffern vorgenommen, weil ansonsten nicht freigegebene Änderungen im Systempuffer für optimistische TA sichtbar würden. Da für pessimistische TA das Durchkommen immer garantiert werden soll (außer bei Deadlock), müssen in der Validierung von ihnen gesetzte Sperren berücksichtigt werden. Außerdem müssen sich nun auch pessimistisch synchronisierte TA „validieren" sowie eine Schreibphase vornehmen.

In der „Validierungsphase" einer pessimistisch synchronisierten Änderungs-TA wird zunächst (wie bei FOCC, s.o.) deren Write-Set auch für weitere Zugriffe optimistischer TA blockiert. Damit wird bei Kombination mit FOCC verhindert, daß sich der relevante Read-Set-Anteil aktiver TA ändert, und bei Kombination mit BOCC bzw. BOCC+ können dadurch vermeidbare Rücksetzungen eingespart werden. Danach werden alle aktiven optimistischen TA zurückgesetzt, die auf ein zu änderndes Objekt zugegriffen haben. Bei Kombination mit BOCC+ erwirbt die pessimistische TA zusätzlich noch ihre TA-Nummer und vermerkt diese als Zeitstempel in den geänderten Objekten sowie in der Objekttabelle. In der Schreibphase werden für pessimistische TA nach dem Logging und dem Einbringen der Änderungen noch die Sperren freigegeben bzw. die Blockierungen für optimistische TA aufgehoben sowie ggf. wartende TA aktiviert.

Daneben ist auch der Validierungstest für eine optimistische TA T_j zu erweitern, weil T_j keine Objekte ändern darf, die eine aktive pessimistische TA bereits gelesen hat. Daher ist bei der Validierung zusätzlich zu überprüfen, ob eine laufende pessimistische TA eine Sperre für ein Objekt aus WS_j hält; wenn ja, muß T_j zurückgesetzt werden. Auch hierbei muß vor der Validierung WS_j blokkiert werden, um zu verhindern, daß sich die relevante Sperrmenge aktiver TA während der Validierung ändert. Die Validierungen werden zwar seriell ausgeführt, die Schreibphasen können jedoch parallel erfolgen. Eine genauere Beschreibung der Algorithmen findet sich in [27].

Durch die Bevorzugung pessimistischer TA ist natürlich das Rücksetzrisiko für optimistische TA größer geworden, so daß im allgemeinen nur noch für kurze TA bzw. in konfliktarmen Anwendungen eine optimistische Synchronisation anzuraten ist. Ein zyklischer Restart kann jedoch leicht vermieden werden, indem sich eine gescheiterte TA bei der zweiten Ausführung pessimistisch synchronisiert.

3. Optimistische Synchronisation in VDBS mit zentraler Validierung

Verteilte Datenbanksysteme bestehen aus einer Menge autonomer Rechner, die lose miteinander gekoppelt sind, d.h. ausschließlich über Nachrichten miteinander kommunizieren. Dabei ist die

Plattenperipherie, auf der die Daten der Datenbank gespeichert sind, strikt partitioniert, so daß jede Platte genau einem Rechner (primär) zugeordnet ist. Die Daten selbst dagegen sind entweder ebenfalls partitioniert (keine Redundanz) oder teilweise oder vollständig repliziert an jedem Rechner verfügbar [28]. Die Rechner können entweder geographisch verteilt oder in räumlicher Nachbarschaft angeordnet sein.

Gemäß der in [29] vorgestellten Klassifikation von Mehrrechner-Datenbanksystemen fallen VDBS wegen der Partitionierung der Plattenperipherie in die Klasse der sogenannten DB-Distribution-Systeme. Im Gegensatz dazu stehen DB-Sharing-Systeme, bei denen jeder Rechner direkt auf alle Platten (Daten) zugreifen kann. DB-Distribution und DB-Sharing werden dabei als die geeignetsten Architekturtypen zur Realisierung von Hochleistungs-Datenbanksystemen angesehen, die folgende Hauptanforderungen zu erfüllen haben:

- Abwicklung hoher Transaktionsraten,
- „permanente" Verfügbarkeit,
- modulare Erweiterungsfähigkeit,
- einfache Handhabbarkeit.

Diese Anforderungen werden in [29] ausführlich diskutiert; ein Vergleich zwischen DB-Distribution und DB-Sharing findet sich in [29, 30].

Obwohl die meisten Veröffentlichungen über optimistische Synchronisationsverfahren auf zentralisierte DBS beschränkt sind, ist in letzter Zeit ein verstärktes Interesse an einer Übertragung der Verfahren auf verteilte DBS zu beobachten. Als Gründe dafür werden dabei vor allem folgende Vorteile des optimistischen Ansatzes im Vergleich zu Sperrverfahren angeführt: eine höhere Parallelität, eine einfache TA- und Crash-Recovery (UNDO) sowie die Deadlock-Freiheit (für rein optimistische Verfahren), womit die aufwendige Erkennung globaler Deadlocks entfällt. Bei einigen Verfahren (z. B. bei zentraler Validierung) bzw. bei Anwendung einiger Optimierungsmöglichkeiten (Kap. 5) ergeben sich zudem Kommunikationseinsparungen im Vergleich zu Sperrverfahren. Allerdings werden durch die Mehrrechner-Architektur auch zusätzliche Probleme eingeführt, die für optimistische Verfahren einer Lösung bedürfen. So wird die Gewährleistung der **globalen Serialisierbarkeit** durch die verteilte Ausführung von TA erschwert, da hierzu die lokale Serialisierbarkeit in jedem Knoten zwar notwendig, jedoch keineswegs hinreichend ist. Die Validierung einer TA ist außerdem unter Umständen mit Kommunikation verbunden, wodurch eine zeitliche Trennung von Validierungs- und Schreibphase entsteht, was zusätzliche Probleme bereitet.

In einigen Vorschlägen [31–33] wurde zur Gewährleistung der globalen Serialisierbarkeit das **explizite Führen eines globalen Abhängigkeitsgraphen,** entweder an zentraler Stelle oder verteilt (ggf. repliziert an allen Knoten), vorgesehen. Bei der Validierung einer TA ist dann im wesentlichen zu überprüfen, ob die validierende TA in einen Zyklus verwickelt ist oder nicht. Damit kann die Anzahl der Rücksetzungen prinzipiell auf ein Minimum beschränkt werden, da keine weiteren Beschränkungen wie z. B., daß jeweils die aktuellsten Änderungen gesehen wurden, vorliegen. Allerdings ist bereits im zentralen Fall die Zyklensuche sowie das Führen eines Abhängigkeitsgraphen äußerst aufwendig, da während der Ausführung einer TA eine Anpassung meist häufig erforderlich ist. Weiterhin besteht die Gefahr, daß der Graph „explodiert", da auch die Abhängigkeiten zu bereits beendeten TA protokolliert werden müssen. Diese Probleme verschärfen sich im verteilten Fall erheblich, da ein globaler Abhängigkeitsgraph nicht nur wesentlich größer ist, sondern da wegen der Entkopplung der Rechner seine Konstruktion noch um eine Größenordnung aufwendiger wird (Kommunikation). So zeigt auch eine genauere Analyse der Algorithmen, daß der Kommunikationsaufwand mit diesen Verfahren höher ist als mit anderen, noch vorzustellenden Alternativen [27]. Aus diesen Gründen wird das explizite Führen eines globalen Abhängigkeitsgraphen hier nicht weiter betrachtet.

Vielmehr soll in dieser Arbeit vorwiegend untersucht werden, wie die optimistischen Verfahren aus Kap. 2 auf VDBS übertragen werden können. Die einfachste Lösung der neu zu behandelnden Schwierigkeiten wird dabei möglich, wenn sämtliche Validierungen auf einem zentralen Knoten durchgeführt werden. Die Realisierung eines solchen zentralen Ansatzes wird daher in diesem Abschnitt zuerst beschrieben; in Kap. 4 folgt dann die Darstellung von verteilten optimistischen Synchronisationsverfahren.

Bei allen Verfahren für VDBS wird generell unterstellt, daß jedes Objekt der Datenbank nur an einem Knoten gespeichert ist (keine Replikation) und daß zum Zugriff auf Daten eines anderen Rechners in diesem eine Sub-TA gestartet wird. Eine TA, die nur Objekte des Rechners, an dem sie gestartet wurde, referenziert, wird als **lokale TA** bezeichnet, anderenfalls als **globale TA.** Jede globale TA besteht aus einer Primär-TA, die am Ursprungsknoten der TA ausgeführt wird, sowie einer Menge von Sub-TA, die von der Primär-TA initiiert werden. Die Sub-TA einer TA können parallel ausgeführt werden; zu einer TA soll jedoch pro Rechner höchstens eine Sub-TA existieren, selbst wenn mehrere Aufträge für die TA an dem Rechner bearbeitet werden.

Validierung auf einem zentralen Knoten

Bei diesem Ansatz werden alle Validierungen an einem festen Knoten durchgeführt, der im folgenden als zentraler Validierungsknoten (ZVK) bezeichnet wird. Sinnvollerweise werden auf diesem Knoten nur die Validierungen vorgenommen (und keine TA-Verarbeitung), um eine Überla-

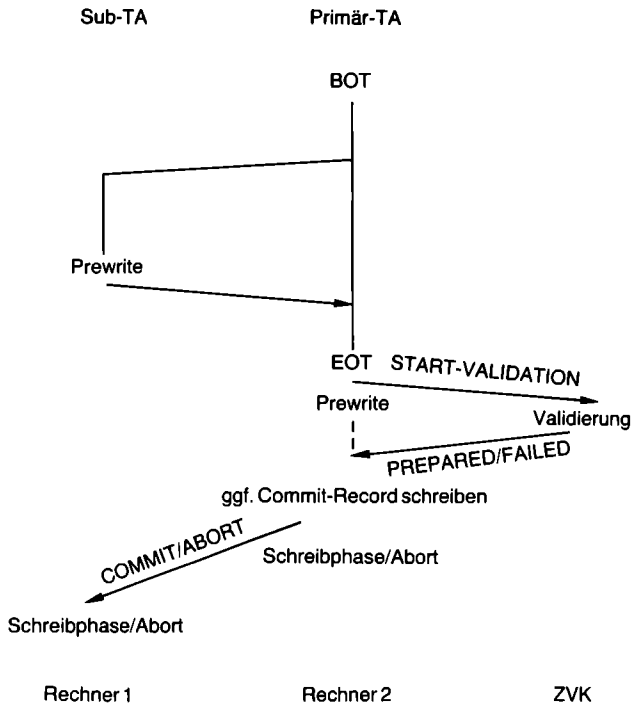

Bild 2. Zentrale Validierung in VDBS (Beispiel).

stung zu vermeiden und schnelle Bedienzeiten zu ermöglichen.

Der Ablauf von TA-Verarbeitung und der Synchronisation mit dem ZVK ist in Bild 2 für ein einfaches Beispiel dargestellt.

Damit am Ende einer globalen TA die Validierung auf dem ZVK angestoßen werden kann, sendet jede Sub-TA nach ihrer Ausführung, zusammen mit der ohnehin notwendigen Fertigmeldung (mit der auch Ergebnisse zurückgeliefert werden), ihren Read- und Write-Set an die Primär-TA. Zuvor sichert jedoch die Sub-TA die beabsichtigten Änderungen auf die lokale Log-Datei (prewrite). Am Ende der Primär-TA wird dann der vollständige Read- und Write-Set zur Validierung an den ZVK geschickt, der das Ergebnis der Validierung der Primär-TA zurückliefert. Bei positivem Ausgang schreibt diese noch nicht gesicherte Änderungen sowie den Commit-Record auf die Log-Datei. Anschließend teilt sie das Ergebnis allen Sub-TA mit, die Änderungen vorgenommen haben, so daß diese sich entweder zurücksetzen bzw. die Schreibphase vornehmen können. Da Sub-TA bereits in der Lesephase alle Änderungen gesichert haben, erspart man sich (wie beim verteilten Zweiphasen-Commit-Protokoll üblich) eine explizite Aufforderung zum Schreiben der Log-Daten bei TA-Ende. Damit werden sowohl Kommunikationsvorgänge eingespart sowie eine schnellere EOT-Behandlung ermöglicht. Die TA gilt als (erfolgreich) beendet, sobald der Commit-Record an ihrem Ursprungsknoten auf dem Log steht.

Diese Beschreibung zeigt, daß zur Synchronisation einer TA nur eine Nachricht (die zur Validierung) erforderlich ist, auf die synchron gewartet werden muß. Bei einem Sperrverfahren mit zentralem Lock-Manager [34] dagegen erfordert jede Sperranforderung eine synchrone Nachricht, was einen wesentlich höheren Kommunikationsoverhead verursacht. Der Vorteil des geringen Nachrichtenbedarfs wird allerdings durch die Tatsache abgeschwächt, daß zur Validierung im ZVK nur eine **BOCC-artige Synchronisation** in Betracht kommt. Die FOCC-Alternative scheitert daran, daß es unmöglich ist, in einem Rechner die aktuellen Read-Sets der in den verschiedenen Knoten laufenden TA zu kennen. Daher kommt zur Synchronisation in erster Linie der BOCC+-Ansatz in Frage, der nahezu unverändert übernommen werden kann. Im ZVK wird dazu der globale TA-Zähler TNC, der bei jeder (erfolgreichen) Validierung inkrementiert wird, geführt sowie (zur Validierung) die Objekttabelle mit den Zeitstempeln für geänderte Objekte. Die zeitliche Trennung zwischen Validierungs- und Schreibphase bereitet hier keine neuen Probleme, weil über die Zeitstempel, die auch in den Objekten selbst gespeichert werden, immer erkannt wird, falls eine mittlerweile veraltete Objektversion gelesen wurde. Ein frühzeitiges Zurücksetzen zum Scheitern verurteilter TA zum Einsparen unnötiger Arbeit ist auch hier noch möglich. Dazu werden nach der erfolgreichen Validierungsnachricht vom ZVK (PREPARED-Meldung) auf den Rechnern, auf denen die validierte TA Änderungen einbringen will, laufende TA zurückgesetzt, die auf eines der zu schreibenden Objekte zugegriffen haben.

Damit bleiben alle Vorteile des BOCC+-Verfahrens (siehe 2.3) auch hier erhalten. Von besonderer Bedeutung ist dabei die sehr einfache und effizient realisierbare Form der Validierung von BOCC+, um auch bei hohen TA-Raten einen Engpaß zu vermeiden. Denn der Validierungsaufwand pro TA ist hier unabhängig von der Anzahl der Rechner oder der eingestellten Parallelität; es sind für eine Validierung nur ein Vergleich pro Read-Set-Element und nur eine Änderung der Objekttabelle pro Write-Set-Element notwendig. Damit läßt sich durch einen ausreichenden leistungsstarken ZVK (z. B. 20–30 MIPS) ein Engpaß auch bei sehr hohen TA-Raten, etwa 1000 kurzen TA/sec, vermeiden, trotz der notwendigen Wartung der Datenstrukturen sowie des Kommunikations-Overheads [35].

Dennoch ist dieses Verfahren auf konfliktärmere Anwendungen beschränkt, weil (wie schon für den zentralen Fall ausgeführt) ansonsten mit zu vielen Rücksetzungen sowie zyklischen Restarts zu rechnen ist. Daher ist auch für Mehrrechner-DBS, für die aufgrund der höheren TA-Raten eher größere Konfliktwahrscheinlichkeiten zu erwarten sind, eine Kombination mit Sperrverfahren in die Überlegungen einzubeziehen. Dies ist analog zu den Ausführungen in 2.4 relativ einfach realisierbar, indem die Objekttabelle im ZVK um die ent-

sprechenden Sperrinformationen erweitert wird. Allerdings wird nun für pessimistisch synchronisierte TA ein hoher Kommunikationsaufwand eingeführt, da jede Sperranforderung an den zentralen Knoten zu richten ist. Eine attraktivere Alternative stellt daher möglicherweise ein sogenannter **Preclaiming-Ansatz** dar, ähnlich wie in [20, 36] vorgeschlagen, mit dem zumindest für kurze TA ein zyklischer Restart vermieden werden sollte. Dabei räumt der ZVK einer (mehrfach) gescheiterten TA einen Sonderstatus ein, indem für den Read- und Write-Set der letzten (erfolglosen) TA-Ausführung ein „Preclaiming" durchgeführt wird. Dies bedeutet, daß dieser Read- und Write-Set bei allen folgenden Validierungen zu berücksichtigen ist, wobei sich im Konfliktfall die validierenden TA zurücksetzen müssen. Damit kann eine erfolgreiche Wiederausführung der gescheiterten TA garantiert werden, sofern keine zusätzlichen Objekte referenziert werden, was für kurze TA relativ wahrscheinlich ist. Für erstmals referenzierte Objekte muß wiederum eine Validierung erfolgen.

Offensichtlich entspricht das Preclaiming dem Sperren von Objekten, wobei sinnvollerweise wieder zwischen lesendem und schreibendem Zugriffswunsch unterschieden wird. So können auch beim Preclaiming Verzögerungen auftreten, wenn mehrere TA für das gleiche Objekt unverträgliche „Sperren" anfordern; Deadlocks können jedoch vermieden werden, weil alle Anforderungen auf einmal gestellt werden. Der Hauptvorteil gegenüber einem „echten" Sperrverfahren liegt in dem stark verringerten Kommunikationsaufwand, weil nicht mehr jede Sperre einzeln angefordert werden muß.

4. Optimistische Verfahren mit verteilter Validierung

Obwohl die Validierung auf einem zentralen Knoten eine relativ einfache Lösung erlaubt bei wahrscheinlich vielfach ausreichender Leistungsfähigkeit, sehen die bisherigen Vorschläge für optimistische Synchronisationsverfahren in verteilten DBS [18, 37–43] fast durchweg eine verteilte Validierung vor. Ein Grund dafür liegt sicher in der Sonderrolle des ZVK, der eine besondere Ausfallbehandlung erfordert („single point of failure") und die Autonomie der anderen Rechner (eine Eigenschaft, die vor allem in ortsverteilten Systemen wesentlich ist) einschränkt. Ein weiterer Nachteil besteht darin, daß selbst lokale TA auf dem ZVK validieren müssen und somit Kommunikation verursachen.

Diese Schwächen sollen mit einem verteilten Validierungsschema, bei dem jede (Sub-)TA an ihrem ausführenden Rechner validiert wird, vermieden werden. In diesem Fall wird dann für rein lokale TA, die idealerweise den größten TA-Anteil ausmachen, keine Interprozessor-Kommunikation erforderlich. Für globale TA ergeben sich allerdings, wie wir gleich sehen werden, eine Reihe von neu zu lösenden Schwierigkeiten. Ein Pluspunkt bei verteilter Validierung ist sicher, daß anstelle einer BOCC-artigen Synchronisation auch die FOCC-Strategie anwendbar ist. Wenn dabei eine Sub-TA bereits während ihrer Lesephase (zur Vermeidung unnötiger Arbeit) zurückgesetzt werden soll, kann sie sich entweder selbständig neu aktivieren oder aber durch eine Nachricht an die Primär-TA das Zurücksetzen der gesamten TA veranlassen (z. B. nach mehrmaligem Scheitern).

In den meisten der oben angeführten Literaturstellen wird ein **zweiphasiges Validierungsschema** für globale TA vorgesehen, bei dem ein Zweiphasen-Commit-Protokoll mit durchgeführt wird und mit dem aus den lokalen Validierungsergebnissen ein gemeinsames Resultat (Commit oder Abort) erzielt werden soll. Die Verarbeitung einer TA sieht vor, daß jede Sub-TA ihr Ende der Primär-TA mitteilt. Wenn die TA vollständig bearbeitet ist, sendet die Primär-TA an alle Knoten, auf denen eine Sub-TA ausgeführt wurde, eine Validierungsaufforderung (START-VALIDATION); danach unterzieht sie sich am Ursprungsknoten selbst einer lokalen Validierung. Nach einer erfolgreichen lokalen Validierung sichert jede Sub-TA die von ihr beabsichtigten Änderungen (prewrite) und schickt eine PREPARED-Nachricht an die Primär-TA; bei gescheiterter Validierung setzt sich die Sub-TA zurück, und es wird eine FAILED-Nachricht gesendet. Waren alle lokalen Validierungen erfolgreich, so verschickt die Primär-TA (nach Schreiben des Commit-Records) eine COMMIT-Nachricht an die betroffenen Rechner, um die Schreib-

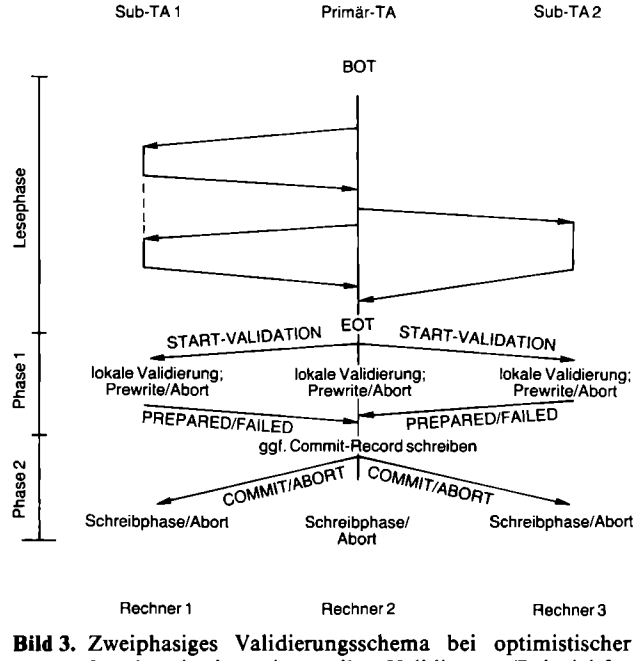

Bild 3. Zweiphasiges Validierungsschema bei optimistischer Synchronisation mit verteilter Validierung (Beispiel für eine TA mit zwei Sub-TA).

phasen der Sub-TA anzustoßen; anderenfalls wird durch eine ABORT-Nachricht das Zurücksetzen der TA eingeleitet. Die durch diese Vorgehensweise entstehenden Abschnitte im Laufe einer TA werden durch Bild 3 noch einmal verdeutlicht. Offenbar erfordert das Verfahren in etwa den gleichen Kommunikationsaufwand wie ein verteiltes Zweiphasen-Sperrprotokoll ohne globale Deadlock-Erkennung. (Hier brauchen lesende Sub-TA nur die Sperren freizugeben, jedoch keine Log-Daten zu schreiben. Bei optimistischer Validierung müssen diese Sub-TA zwar validieren, brauchen jedoch keine Schreibphase vorzunehmen.)

In [18, 37, 41] wird ein etwas abgewandeltes Verarbeitungsmodell verwendet, bei dem jede Sub-TA die lokale Validierung unmittelbar am Ende ihrer Lesephase startet, ohne vorherige Aufforderung durch die Primär-TA. Dabei wird allerdings stillschweigend angenommen, daß die Primär-TA keine weiteren Anforderungen an einen Rechner stellt, auf dem bereits eine Sub-TA ausgeführt wurde (single invocation interface), da sonst eine erneute Sub-TA ausgeführt werden müßte, die u. U. mit früheren Sub-TA derselben globalen TA in Konflikt geraten könnte. Diese Vereinfachung erscheint jedoch nicht gerechtfertigt, da vor allem bei längeren TA durchaus mehrere Abarbeitungen auf demselben Rechner notwendig werden können (in Bild 3 z. B. für Sub-TA 1). Aus diesen Gründen soll hier für die weiteren Überlegungen das oben eingeführte, allgemeinere Verarbeitungsmodell vorausgesetzt werden.

An den nächsten beiden Abschnitten wird zunächst für BOCC bzw. BOCC+ und anschließend für FOCC gezeigt, welche Probleme bei dem verteilten Validierungsschema für eine korrekte Synchronisation noch zu lösen sind und welche prinzipiellen Lösungsmöglichkeiten existieren. In 4.3 wird dann noch kurz auf eine Kombination mit Sperrverfahren eingegangen.

4.1 Grundlegende Probleme und Lösungsmöglichkeiten

Mit dem oben angeführten zweiphasigen Validierungsschema wird trotz der verteilten Validierungen eine eindeutige Entscheidung über den TA-Ausgang möglich, da alle Teilergebnisse am Ursprungsknoten einer TA zusammengefaßt werden. Weiterhin erlaubt es, die Atomizität der TA sicherzustellen, weil nach Phase 1 alle beabsichtigten Änderungen durch das Prewrite gesichert sind. Damit können die Modifikationen einer erfolgreich validierten TA, für die der Commit-Record am Ursprungsknoten auf dem Log steht, selbst bei Rechnerausfällen nachvollzogen werden; ein UNDO ist ohnehin unproblematisch.

Trotzdem müssen für eine korrekte Synchronisation noch Lösungen für die folgenden beiden, eng zusammenhängenden Probleme gefunden werden (siehe auch [44]):

a) Selbst wenn alle lokalen Validierungen einer globalen TA erfolgreich sind, können dennoch nichtserialisierbare Verarbeitungsfolgen entstehen. Eine Lösung des Problems wird durch eine geeignete Koordinierung der lokalen Validierungen möglich.
b) Die zeitliche Trennung zwischen lokaler Validierung und Schreibphase für globale TA erfordert eine Sonderbehandlung für sogenannte unsichere Änderungen. Dies sind Objekte, für die zwar ein Prewrite vorgenommen wurde, für die jedoch noch nicht bekannt ist, ob die beabsichtigte Änderung eingebracht werden kann oder nicht.

Bei der nun folgenden näheren Betrachtung dieser Probleme sowie möglicher Lösungswege gehen wir zunächst von einer BOCC-artigen Synchronisation aus. Besonderheiten bei einer FOCC-Strategie werden dann in 4.2 besprochen.

4.1.1 Koordinierung lokaler Validierungen

Ein einfaches Beispiel für das zugrundeliegende Problem zeigt Bild 4 für zwei globale TA T1 und T2 mit Ursprungsknoten R1 bzw. R2. Alle lokalen Validierungen verlaufen erfolgreich, da die jeweiligen Sub-TA lokal serialisierbar sind. Dennoch besteht im globalen Abhängigkeitsgraphen ein Zyklus zwischen T1 und T2, da T1 ein von T2 gelesenes Objekt ändern will und umgekehrt. Die Entstehung solcher Zyklen ist auf Grund von transitiven Abhängigkeiten natürlich auch bei mehr als zwei beteiligten Rechnern möglich, da lokale Serialisierbarkeit auf jedem Knoten offenbar nicht hinreichend ist für globale Serialisierbarkeit. Die Lösung dieses Problems wurde in [18] und [41] außer acht gelassen!

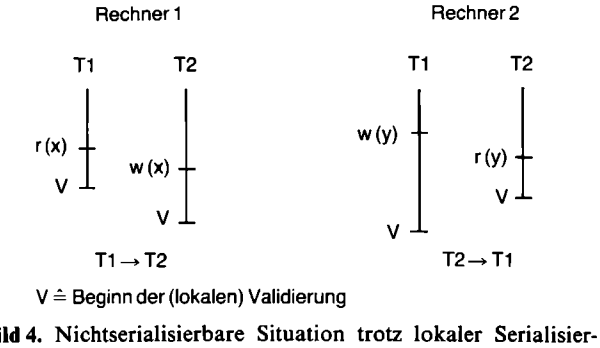

Bild 4. Nichtserialisierbare Situation trotz lokaler Serialisierbarkeit.

Das Problem wird sicherlich umgangen, wenn – wie in Kap. 3 – alle Validierungen an zentraler Stelle durchgeführt werden. Eine andere (zu aufwendige) Lösungsmöglichkeit besteht im expliziten Führen eines globalen Abhängigkeitsgraphen, der dann in der Validierungsphase auf Zyklen durchsucht wird (s. o.). Eine dritte Strategie schließlich ist eine „geeignete" Koordinierung der lokalen Validierungen, wofür wiederum mehrere Möglichkeiten in Betracht kommen:

1. **Erzwingen der gleichen Validierungsreihenfolge auf allen Knoten**
In diesem Fall kommt in obigem Beispiel nur die zuerst validierende TA durch, da für die zweite auf einem der Rechner festgestellt wird, daß die Änderung der zuvor validierten TA nicht gesehen wurde. Die Erzwingung der gleichen Validierungsreihenfolge für globale TA auf allen Rechnern sichert die globale Serialisierbarkeit, wenn die lokale Serialisierungsreihenfolge der lokalen (und hier auch globalen) Validierungsreihenfolge entspricht.

Daß Sub-TA globaler TA auf allen Knoten in der gleichen Reihenfolge validiert werden, kann wiederum durch verschiedene Möglichkeiten realisiert werden:

a) Die einfachste Realisierung ist möglich bei lokaler Anordnung der Rechner und einem Broadcast-Medium (z. B. Bus) zur Rechnerkopplung. Hier stößt die Primär-TA alle lokalen Validierungen mittels einer Broadcast-Nachricht an; die Validierungen brauchen dann nur in der Empfangsreihenfolge dieser Nachrichten vorgenommen zu werden.
b) Jede TA bekommt bei EOT an ihrem Ursprungsknoten eine eindeutige Nummer (z. B. aus Rechner-ID und lokaler Uhrzeit), die ihre Position in der globalen Serialisierungsreihenfolge festlegt. Diese Nummer wird bei den Validierungsaufforderungen mitgeschickt, so daß in jedem Knoten die gleiche Validierungsreihenfolge eingehalten werden kann. Zu spät eintreffende TA (d. h. TA, deren Nummer kleiner ist als die der zuletzt lokal validierten TA) werden zurückgesetzt.
c) Bei EOT holen sich globale TA eine eindeutige TA-Nummer auf einem ausgezeichneten Knoten, der einen globalen TA-Zähler führt. Dieser Vorschlag aus [9] bringt jedoch offensichtlich keine Vorteile gegenüber Lösung b); vielmehr werden zusätzliche Kommunikationen verursacht.
d) Die lokalen Validierungen werden über ein Token-Ring-Konzept koordiniert, bei dem an einem Rechner Validierungen nur bei Token-Besitz vorgenommen werden dürfen. Mit dem Token wird eine Liste von (globalen) TA mitgeführt, für die eine lokale Validierung vorzunehmen ist, sofern eine Sub-TA am jeweiligen Knoten ausgeführt wurde. Da diese Liste in jedem Knoten in der gleichen Reihenfolge abgearbeitet wird, ergibt sich die gewünschte Gleichheit der Validierungsreihenfolge. Das Ergebnis einer globalen Validierung steht dann stets nach einem Ringumlauf fest. Ein offensichtlicher Nachteil des Verfahrens ist, daß die mögliche Parallelität bei der Durchführung lokaler Validierungen nicht genutzt wird, vielmehr bestimmt die Gesamtanzahl der Rechner (und nicht die Anzahl der Rechner, auf denen Sub-TA ausgeführt wurden) die Zeitdauer, bis das Ergebnis aller Validierungen vorliegt. Mit zunehmender Rechneranzahl verschlechtern sich nicht nur die Antwortzeiten wegen der längeren Umlaufzeiten; es wird auch immer schwieriger, die zunehmende Anzahl von Validierungen wegen der fehlenden Parallelität abzuarbeiten. Ein Vorteil ist die erheblich geringere Gesamtanzahl der zur globalen Validierung benötigten Nachrichten, verglichen mit den Alternativen a–c. Die Verwendung einer Token-Ring-Topologie bei optimistischer Synchronisation wird in [45] genauer diskutiert.

In allen vier Fällen ist es möglich, wie für BOCC bzw. BOCC+ erforderlich, einen globalen Zähler zu führen, der zur Ableitung einer global eindeutigen TA-Nummer benutzt wird.

2. **Verwendung von Zeitstempeln nach [40]**
Hierbei wird, wie bei obiger Alternative 1b, die Position einer TA in der globalen Serialisierungsreihenfolge durch einen systemweit eindeutigen Zeitstempel festgelegt, der bei EOT zugewiesen wird. Jedoch brauchen hier die lokalen Validierungen nicht unbedingt in Zeitstempelreihenfolge vorgenommen zu werden, sondern es wird bei ihnen für eine TA T_j ggf. überprüft, ob T_j sich vor den bereits validierten TA T_i mit größerem Zeitstempel einreihen läßt (Voraussetzung $WS_j \cap RS_i = \varnothing$); gegenüber TA mit kleinerem Zeitstempel erfolgt eine BOCC-artige Validierung wie für zentralisierte DBS. Um die Read-Sets validierter TA nicht aufheben zu müssen bzw. um für die BOCC-Validierung unnötige Rücksetzungen zu umgehen, wird auch hier vorgesehen, für jedes referenzierte Objekt einen Lese- bzw. Schreibzeitstempel zu führen (= Zeitstempel der TA, die das Objekt zuletzt gelesen bzw. geändert hat). Wenn im Beispiel von Bild 4 der Zeitstempel von T1 kleiner ist als der von T2, dann verlaufen die beiden Validierungen an Knoten 1 erfolgreich. An Knoten 2 jedoch muß bei der Validierung von T1 der Read-Set der „jüngeren" TA T2 beachtet werden, was zur Rücksetzung von T1 führt.

Es ist klar, daß durch die Wahl des Zeitstempels globale TA gegenüber rein lokalen TA benachteiligt sind. Zur Verbesserung wird daher in [40] vorgeschlagen, daß sich die betroffenen Rechner vor der Validierung zunächst auf die Festlegung eines möglichst günstigen Zeitstempels für die TA einigen. Dadurch lassen sich dann zwar möglicherweise Rücksetzungen vermeiden, jedoch auf Kosten zusätzlicher Kommunikationsverzögerungen zur Durchführung der „Verhandlungen".

3. **Warten während der Validierung globaler TA**
Bei diesem Vorschlag nach [37, 39] wird vorgesehen, daß die lokale Validierung einer Sub-TA ggf. so lange verzögert wird, bis alle globalen TA, die an dem Knoten zuvor validiert haben und zu denen eine direkte oder indirekte Abhängigkeit im lokalen Abhängigkeitsgraphen besteht, beendet sind. Damit wird erreicht, daß zyklische Abhängigkeiten zwischen globalen TA zu **„Validierungs-Deadlocks"** führen, bei denen die beteiligten Sub-TA ihre Validierungen nicht beendigen können und ein Abschluß der globalen TA somit nicht möglich ist. Zur Auflösung dieser Deadlocks wird in [37] ein einfacher Timeout-Mechanismus vorgeschlagen, der natürlich zu unnötigen Rücksetzungen bzw. starken Verzögerungen führen kann. In [39] dagegen wird eine explizite Deadlock-Erkennung vorgesehen, wodurch jedoch zusätzlicher Kommunikationsaufwand eingeführt wird. Ein weiterer Nachteil der Verfahrensweise besteht in der Notwendigkeit eines lokalen Abhängigkeitsgraphen, da ansonsten transitive Abhängigkeiten zwischen globalen TA über lokale TA hinweg unentdeckt bleiben könnten.

Bei dem Beispiel in Bild 4 würde die lokale Validierung von T2 an Knoten 1 und die von T1 an Knoten 2 verzögert werden, da eine Lese-Schreib-Abhängigkeit zur bereits validierten Sub-TA von T1 bzw. T2 vorliegt, die globale TA jedoch noch nicht beendet ist. Die Auflösung des Deadlocks zwischen T1 und T2 erfolgt durch Zurücksetzen von mindestens einer der beiden TA.

4.1.2 Behandlung unsicherer Änderungen

Im Gegensatz zu zentralisierten DBS ist bei dem zweiphasigen Validierungsschema das Schicksal einer globalen TA nach erfolgreicher lokaler Validierung auf einem der Knoten völlig ungewiß. Die von lokal erfolgreich validierten Sub-TA, die auch als **semicommitted TA** bezeichnet werden, beabsichtigten Änderungen sind demnach „unsicher", da sie je nach Validierungsausgang der globalen TA weggeworfen oder eingebracht werden. Für die Behandlung dieser unsicheren Änderungen in der Lesephase und bei der Validierung lokaler TA kommen im wesentlichen folgende Alternativen in Betracht:

a) Sperren unsicherer Änderungen.
Bei der ersten Möglichkeit wird der Zugriff auf unsichere Änderungen blockiert, bis der Ausgang der globalen TA feststeht. Damit lassen sich unnötige Rücksetzungen vermeiden, falls darauf bestanden wird, daß die lokale Serialisierungsreihenfolge der lokalen Validierungsreihenfolge entspricht; allerdings wird durch die Blockierungen die Parallelität reduziert. Deadlocks können nicht entstehen, da die TA, auf die gewartet wird, ihre Lesephase bereits beendet hat und somit nicht mehr auf solche Sperren laufen kann. Dieses Sperren der Write-Sets von semicommitted TA entspricht dem Blockieren der Write-Sets während der Schreibphase im zentralen Fall (2.2). Aufgrund der Kommunikation sind jetzt allerdings die Verzögerungen entsprechend größer; außerdem waren sie umsonst, falls die globale TA zurückgesetzt werden muß.

Ein großer Vorteil des „Sperrens" unsicherer Änderungen ist, daß bei BOCC-artiger Synchronisation Validierung und Schreibphasen quasi wie im zentralen Fall erfolgen können, wobei insbesondere die Schreibphasen parallel ausgeführt werden können (da durch die Validierung sichergestellt ist, daß nur disjunkte Objektmengen geschrieben werden).

b) Zugriff auf die ungeänderte Version.
Eine andere Möglichkeit besteht darin, beim Zugriff auf ein Objekt, für das eine semicommitted TA eine Änderung beabsichtigt, stets die aktuell gültige, ungeänderte Version zur Verfügung zu stellen. Dies ist allerdings wenig sinnvoll, wenn (wie üblicherweise der Fall) sich die laufenden TA in der Serialisierungsreihenfolge nach allen bereits (lokal) validierten TA einreihen müssen. Denn in diesem Fall wird die auf eine alte Objektversion zugreifende TA zurückgesetzt, wenn die globale TA erfolgreich abschließt, wovon optimistischerweise auszugehen ist. Der Zugriff auf die ungeänderte Objektversion kommt daher allenfalls in Frage, wenn sich die betreffende TA in der Serialisierungsreihenfolge vor der die Änderung beabsichtigenden globalen TA einreihen läßt. Die Entscheidung darüber ist – wie in [27] näher ausgeführt – im verteilten Fall i.a. sehr komplex und soll hier nicht weiter ausgeführt werden. Eine elegante Lösung für Lese-TA unter Verwendung von Zeitintervallen wird in 5.1 beschrieben; allerdings müssen dabei auch alle Schreibphasen in der Serialisierungsreihenfolge ausgeführt werden, wodurch vor allem für lokale Änderungs-TA erhebliche Verzögerungen entstehen können.

c) Zugriff auf die unsichere Änderung.
Bei dieser (sehr optimistischen) Variante wird davon ausgegangen, daß die globale TA erfolgreich validieren kann. Daher wird sofort nach erfolgreicher lokaler Validierung der Zugriff auf die ungesicherten Änderungen freigegeben, um eine möglichst hohe Parallelität bei der TA-Verarbeitung zu erreichen. Allerdings machen auf unsichere Änderungen zugreifende TA ihr Schicksal abhängig von dem der globalen TA, deren Änderungen gesehen wurden. Sinnvollerweise warten dann diese „abhängigen" TA vor ihrer lokalen Validierung ab, ob ihr Optimismus berechtigt war, d.h., ob die globalen TA, deren Änderungen gesehen wurden, erfolgreich waren oder nicht (Deadlocks können auch hier-

bei nicht auftreten). Würden nämlich diese abhängigen TA ebenfalls wieder unsichere Änderungen zugänglich machen, könnte eine Rücksetzung einen Domino-Effekt nach sich ziehen. Der Zugriff auf unsichere Änderungen wird u.a. in [38, 39] vorgesehen, soll hier jedoch nicht weiter verfolgt werden.

Eine Kombination der Alternativen a) bis c) ist möglich.

4.2 Besonderheiten bei FOCC

Wie in Kapitel 2 schon ausgeführt, besitzt der FOCC-Ansatz, bei dem gegen noch laufende TA validiert wird, mehr Möglichkeiten zur Konfliktbehebung und zur Vermeidung von zyklischen Restarts als die BOCC-Verfahren. Für VDBS ergibt sich als weiterer Vorteil, daß beim Zurücksetzen einer Sub-TA während einer Lesephase nicht unbedingt die gesamte TA abgebrochen werden muß; sinnvoller ist es nämlich u.U., nur die Sub-TA neu zu starten. Außerdem erhöhen sich die Validierungschancen einer Sub-TA, die ihre Lesephase ohne Rücksetzung überstanden hat, da sie sich dann bereits gegen die in dieser Zeit validierenden Änderungs-TA „abgesichert" hat. Natürlich sind aber auch bei FOCC-artiger Validierung die im letzten Abschnitt angesprochenen Probleme einer verteilten Validierung zu lösen, wobei im wesentlichen dieselben Lösungen anwendbar sind. Durch die vorwärtsorientierte Validierung ergeben sich jedoch, verglichen mit BOCC, einige Punkte, die nun neu zu behandeln sind.

Validierung lokaler TA

Wir betrachten hierzu nur den Fall, bei dem sich lokale TA an das aktuelle Ende der lokalen Serialisierungsreihenfolge anhängen müssen. Die genannte Voraussetzung besagt, daß eine lokale TA alle Änderungen gesehen haben muß, die von bereits validierten TA durchgeführt bzw. beabsichtigt sind. Dazu reicht, wie Bild 5 zeigt, die übliche FOCC-Validierung, bei der nur Änderungs-TA gegenüber laufenden TA zu validieren haben, nicht mehr aus.

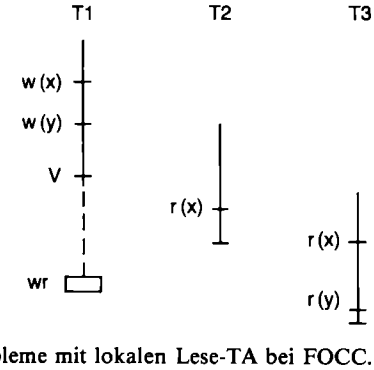

Bild 5. Probleme mit lokalen Lese-TA bei FOCC.

Denn demnach bräuchten sich die beiden lokalen Lese-TA T2 und T3 nicht zu validieren; beide TA haben jedoch nicht alle Änderungen der globalen TA T1 gesehen! Weil dieses Problem durch die semicommitted TA eingeführt wird, ergeben sich analog zu 4.1.2 drei Lösungsmöglichkeiten. Der vielversprechendste Ansatz ist auch hier das Blokkieren unsicherer Änderungen, wobei lokal erfolgreich validierte Sub-TA ihren Write-Set bis nach ihrer Schreibphase (bzw. bis zu ihrer Zurücksetzung) für Zugriffe aktiver TA sperren. In diesem Fall brauchen lokale Lese-TA keine eigene Validierung vorzunehmen, es genügt die übliche FOCC-Validierung. Daneben werden auch hier wieder unnötige Rücksetzungen vermieden.

Für lokale Update-TA ist wie im zentralen Fall der Write-Set mit dem Read-Set aller noch nicht validierten TA zu vergleichen, wobei im Konflikt wie üblich entweder die validierende oder die in der Lesephase befindliche TA zurückgesetzt werden kann. Durch Blockieren der Write-Sets von semicommitted TA ist sichergestellt, daß der Write-Set einer (erfolgreich validierten) lokalen TA disjunkt ist zu den Write-Sets aller semicommitted TA. Daher kann auch hier eine lokale TA ihre Schreibphase sofort durchführen (keine Verzögerung, bis alle in der Serialisierungsreihenfolge vorher kommenden TA beendet sind).

Validierung globaler TA

Um die globale Serialisierbarkeit sicherstellen zu können, gehen wir hier davon aus, daß an jedem Rechner die Validierungen globaler TA in der gleichen Reihenfolge vorgenommen werden (dies kann durch eine der in 4.1.1 genannten Alternativen realisiert werden) und daß sich eine validierende Sub-TA an das Ende der lokalen Serialisierungsreihenfolge anfügen muß. Wie schon erwähnt, ist es aufgrund der letzten Bedingung sinnvoll, den Zugriff auf Objekte zu blockieren, die zum Write-Set einer semicommitted TA gehören, da ansonsten unnötige Rücksetzungen verursacht werden. Da sich die Sub-TA bis zum Ende ihrer lokalen Lesephase wegen FOCC bereits gegen parallele Änderer abgesichert haben, steht nur noch eine Validierung gegenüber den Änderungs-TA T_j aus, die zwischen dem Ende der Lesephase und dem Beginn der Validierungsphase validiert haben. Hierzu kommen folgende zwei Möglichkeiten in Betracht:

a) Die Sub-TA T_i validiert sich gegenüber diesen TA, wobei für alle T_j gelten muß:
$RS_i \cap WS_j = \emptyset$
b) Die TA T_j berücksichtigen in ihrer FOCC-Validierung nicht nur in der Lesephase befindliche TA, sondern auch Sub-TA, die auf den Beginn ihrer Validierung warten.

Dabei ist Lösung b) offenbar vorzuziehen, da sie die freie Wahl des Opfers zuläßt; insbesondere kann damit einer (z.B. schon mehrfach gescheiter-

ten) globalen TA ein Durchkommen zugesichert werden. Bei dieser Methode ist bei der Validierung einer Sub-TA also zunächst sicherzustellen, daß sie nicht schon während der Wartezeit auf die Validierungsaufforderung zum Scheitern verurteilt wurde. Für reine Lese-Sub-TA ist damit die Validierung bereits beendet, während Änderer noch den üblichen FOCC-Test vornehmen, bei dem ihr Write-Set gegen den Read-Set noch nicht validierter TA (inklusive auf die Validierung wartender Sub-TA) verglichen wird. Die Schreibphasen globaler Änderungs-TA brauchen auch hier nicht notwendigerweise in Serialisierungsreihenfolge vorgenommen zu werden. Wie angedeutet, müssen bei FOCC und dem zweiphasigen Validierungsschema auch globale Lese-TA validieren, damit sichergestellt ist, daß sie auf allen Rechnern denselben konsistenten DB-Zustand gesehen haben. Optimierungen, die eine Validierung für Lese-TA vermeiden, werden in Kap. 5 vorgestellt.

4.3 Entwurfsalternativen

Die bisherigen Ausführungen zeigen, daß die einfachsten Lösungen bei verteilter Validierung möglich sind, wenn die Validierungen von globalen TA an jedem Rechner in der gleichen Reihenfolge vorgenommen werden und wenn sich eine validierende TA stets an das Ende der lokalen Serialisierungsreihenfolge anhängen muß. Denn in diesem Fall ist es zur Vermeidung unnötiger Rücksetzungen sinnvoll, die Objekte aus dem Write-Set von semicommitted TA zu sperren, bis das Schicksal der globalen TA feststeht und (im Erfolgsfall) deren Schreibphase beendet ist. Dies wiederum gestattet ähnlich einfache Validierungsregeln wie im zentralen Fall sowie parallele Schreibphasen, die nicht in Serialisierungsreihenfolge vorgenommen zu werden brauchen. Die Blockierungen von Objekten während der Validierungs- und Schreibphasen globaler TA können zwar Parallelitätseinbußen bzw. Antwortzeitverschlechterungen verursachen, jedoch ist die Dauer der Blockierungen i. a. weitaus kürzer als bei einem Sperrverfahren.

Wie bei einem Rechner oder bei zentraler Validierung erscheint es vor allem für die BOCC-Verfahren notwendig, eine **Kombination mit Sperrverfahren** zur Reduzierung von Rücksetzungen und Vermeidung zyklischer Restarts vorzunehmen. Dies ist analog zum zentralen Fall (2.4) möglich, ohne daß sich zusätzliche Kommunikation ergibt. Denn das Starten von Sub-TA ist ohnehin notwendig, und jede Sub-TA kann lokal synchronisiert werden; ein Zweiphasen-Commit ist auch bei rein optimistischer Synchronisation nötig. Allerdings wird jetzt eine Behandlung globaler Deadlocks notwendig. In [41] wurde versucht, den Vorschlag aus [23] auf verteilte DBS zu übertragen, jedoch ist die vorgeschlagene Strategie nicht korrekt, da für optimistisch synchronisierte TA die in 4.1 angesprochenen Probleme nicht behandelt werden.

Weitere Entwurfsalternativen, die sowohl in zentralisierten DBS als auch in VDBS mit zentraler oder verteilter Validierung einsetzbar sind, werden im nächsten Kapitel angesprochen.

5. Optimierungsmöglichkeiten

Bisher wurde bei den meisten Verfahren unterstellt, daß die Validierungsreihenfolge der TA zugleich die Serialisierungsreihenfolge darstellt, so daß eine TA nur erfolgreich validieren konnte, wenn stets die aktuellste Objektversion gesehen wurde. In 4.1.2 wurde aber bereits angesprochen, daß sich Rücksetzungen vermeiden lassen, wenn ein Vorverlegen der TA in der Serialisierungsreihenfolge zugelassen wird. Denn in diesem Fall kann eine Rücksetzung umgangen werden, auch wenn mittlerweile veraltete Objektversionen gesehen wurden, sofern diese einem TA-konsistenten DB-Zustand entsprechen. Die im folgenden beschriebene Realisierungsform dieser Methode mittels Zeitintervallen ist sowohl für zentralisierte DBS als auch für VDBS mit zentraler und verteilter Validierung anwendbar. Eine andere Realisierungsmöglichkeit dieser Optimierung, die aber primär auf zentralisierte Systeme beschränkt ist, findet sich in [9, 46]. In 5.2 und 5.3 werden dann noch zwei weitere Optimierungsmöglichkeiten für zentralisierte DBS und VDBS vorgestellt: der Einsatz eines Multiversion-Konzeptes und die Beschränkung auf Konsistenzebene 2. Dabei ist das Durchkommen für Lese-TA sogar immer sichergestellt, so daß sich eine erhebliche Reduzierung der Rücksetzrate erzielen läßt. Die Gefahr zyklischer Restarts wird damit gleichzeitig auch stark verringert.

5.1 Vorverlegen von TA in der Serialisierungsreihenfolge

Die hier vorgestellte Realisierungsform wurde aus [42] abgeleitet und verwendet Zeitintervalle, um für Lese-TA ein Vorverlegen in der Serialisierungsreihenfolge vor bereits validierte TA zu ermöglichen. Änderungs-TA sollen sich dagegen nach wie vor an das aktuelle Ende der (globalen) Serialisierungsreihenfolge anhängen, indem sie sich gemäß einem der vorgestellten Verfahren synchronisieren. Neben der Reduzierung des Rücksetzrisikos für Lese-TA werden sich in VDBS für diese TA auch starke Kommunikationseinsparungen ergeben, weil bei EOT ihr Schicksal bereits unmittelbar feststeht (keine Validierung). Es braucht also weder eine Nachricht mit einem zentralen Validierungsknoten ausgetauscht zu werden, noch ist ein zweiphasiges Validierungsschema erforderlich!

Bei der Realisierung wird vorausgesetzt, ähnlich wie beim BOCC+-Verfahren, daß erfolgreiche Änderungs-TA eine eindeutige TA-Nummer erhalten, die ihre Position in der globalen Serialisierungsreihenfolge kennzeichnet. Diese TA-Num-

mer wird zugleich bei den geänderten Objekten als Zeitstempel oder Versionsnummer abgelegt. Damit eine Lese-TA an die Position x in der globalen Serialisierungsreihenfolge eingereiht werden kann, müssen offenbar zwei Bedingungen erfüllt sein:

1. Die TA muß alle Änderungen von TA mit TA-Nummer kleiner x gesehen haben.
2. Die TA darf kein Objekt referenziert haben, das von einer TA geändert wurde, deren TA-Nummer größer als x ist.

Die Position innerhalb der Serialisierungsreihenfolge, an die eine Lese-TA eingeordnet werden kann, läßt sich durch Führung eines Zeitintervalls ermitteln, welches die Abhängigkeiten der Lese-TA zu den bereits validierten Änderungs-TA widerspiegelt. Bei BOT der Lese-TA bekommt das Intervall seinen Initialwert $[0,\infty)$; eine Modifikation des Intervalls erfolgt beim Zugriff auf Datenobjekte sowie bei der Schreibphase parallel laufender Änderungs-TA. In VDBS wird beim Start einer Sub-TA dieser das aktuelle Intervall mitgegeben und während ihrer Ausführung weiterentwickelt. Bei paralleler Bearbeitung einer TA auf mehreren Rechnern wird das Gesamt-Intervall bei der Rückmeldung der Sub-TA bei der Primär-TA durch Durchschnittsbildung der einzelnen Intervalle ermittelt. Ist das Intervall bei EOT der Lese-TA nicht leer, dann kann die Lese-TA ohne weitere Validierung erfolgreich beendet werden, da sie sich in dem durch das Intervall festgelegten Bereich in der globalen Serialisierungsreihenfolge einordnen läßt. Die Lese-TA kann umgekehrt auch schon vor EOT zurückgesetzt werden (um unnötige Arbeit zu vermeiden), sobald sich während der Lesephase ein leeres Intervall ergibt.

Die Anpassung der Intervalle während der Laufzeit von Lese-TA geschieht folgendermaßen: Bei jedem Objektzugriff wird die linke Intervallgrenze der lesenden TA auf das Supremum des aktuellen Wertes dieser Intervallgrenze und des Schreibzeitstempels der gelesenen Objektversion gesetzt. Dies geschieht deshalb, weil die Lese-TA in der Serialisierungsreihenfolge nach dem Erzeuger der gesehenen Objektversion stehen muß. Die rechte Intervallgrenze für eine aktive Lese-TA T wird durch Änderungs-TA, die ein Objekt aus dem Read-Set von T ändern wollen, unmittelbar vor der Schreibphase modifiziert (die Schreibreihenfolge muß hier unbedingt der Serialisierungsreihenfolge entsprechen!). Dabei wird die rechte Intervallgrenze auf das Infimum seines aktuellen Wertes und der TA-Nummer des Änderers gesetzt, weil die Lese-TA in der Serialisierungsreihenfolge vor dem Änderer stehen muß, da dessen Änderung nicht gesehen wurde.

Die Verwendung der Zeitintervalle dürfte einige Rücksetzungen für Lese-TA vermeiden, und in VDBS wird der Kommunikationsaufwand für diese TA auf ein Minimum reduziert. Außerdem brauchen jetzt Objekte, für die eine validierte (semicommitted) TA eine Änderung beabsichtigt, nur noch für Änderungs-TA, nicht jedoch für Lese-TA blockiert zu werden, da sich die Leser vor der validierten (semicommitted) TA in der Serialisierungsreihenfolge einordnen können (höhere Parallelität). Der Nachteil ist, daß alle Schreibphasen gemäß der Serialisierungsreihenfolge vorzunehmen sind, was vor allem in VDBS zu spürbaren Verzögerungen führen kann.

Die Verwendung von Zeitintervallen wurde auch in [47] vorgeschlagen, allerdings für ein Sperrverfahren und zur Vermeidung von Deadlocks. Dort wurde das Intervall einer TA bei jedem Konflikt reduziert; ein leeres Intervall führte ebenfalls zu einer Zurücksetzung, da die TA dann möglicherweise in eine Verklemmung verwickelt ist. Nach [42] ist dabei die Bestimmung der Zeitintervalle problematischer als bei dem optimistischen Verfahren, zumal auch Konflikte mit gescheiterten TA zur Rücksetzung führen können.

5.2 Multiversion-Konzept

Bei dem Multiversion-Konzept, das bereits in Kombination mit Sperrverfahren als sehr wirkungsvoll erachtet wird [48], bekommt eine Lese-TA T während ihrer gesamten Laufzeit eine Sicht auf die Datenbank gewährt, wie sie bei BOT gültig war; d.h., Änderungen, die während ihrer Laufzeit vorgenommen wurden, bleiben für T unsichtbar. Um dies zu realisieren, erzeugt jede erfolgreiche Änderungs-TA eine neue Version der modifizierten Objekte; die Versionen werden in einem Versionen-Pool verwaltet. Da unter Zuhilfenahme der Versionen jeder Lese-TA der bei BOT gültige (und konsistente) DB-Zustand zur Verfügung gestellt wird, braucht für diese TA keinerlei Synchronisation mehr durchgeführt zu werden, auch brauchen sich andere TA nicht mehr gegen Lese-TA zu synchronisieren. Änderungs-TA synchronisieren sich untereinander entsprechend dem gewählten optimistischen Verfahren; sie greifen wie bisher immer auf die aktuellste Version der Daten zu. Da sie sich nicht mehr gegen Lese-TA zu validieren haben, reduziert sich sowohl der Validierungsaufwand als auch die Konfliktwahrscheinlichkeit und damit die Anzahl der Zurücksetzungen. Für Lese-TA ergibt sich in VDBS zudem ein verringerter Kommunikationsaufwand (keine Validierung). So haben auch Simulationen [49, 50] für den zentralen Fall ergeben, daß durch Einsatz eines Mehrversionen-Konzeptes enorme Durchsatz- und Antwortzeitverbesserungen erzielt werden können.

Für die Vorteile, die ein Mehrversionen-Konzept bietet, muß jedoch zum einen in Kauf genommen werden, daß Lese-TA (v.a. lange Leser) nicht immer die aktuellsten Daten sehen. Zum anderen ist ein erhöhter Speicherplatzbedarf zur Haltung der Versionen sowie ein zusätzlicher Verwaltungsaufwand für den Versionen-Pool erforderlich.

Techniken zur Versionen-Pool-Verwaltung (Auffinden von Versionen, Freigabe nicht mehr benötigter Versionen etc.) wurden bereits in [48] beschrieben; allerdings wird dabei der Versionen-Pool auf Platte gehalten, und die Versionenhaltung geschieht auf Seitenebene. Zur Reduzierung des E/A-Aufwandes sollte der Versionen-Pool jedoch weitgehend im Hauptspeicher geführt werden; zur Begrenzung der Synchronisationskonflikte für Änderungs-TA wäre zudem ein kleineres Synchronisationsgranulat als die Seite (Satz, Eintrag) wünschenswert. Erste Überlegungen hierzu finden sich in [51].

Zur Übertragung des Multiversion-Ansatzes auf VDBS muß vor allem geklärt werden, welche Versionen einer Lese-TA zur Verfügung zu stellen sind, so daß ihr der bei ihrem Start gültige DB-Zustand angeboten werden kann. Eine Möglichkeit dazu ist (wie bei BOCC+), die global eindeutigen TA-Nummern als Versionsnummern zu verwenden; hier braucht dann für die Lese-TA bei BOT nur der aktuelle Wert des globalen TA-Zählers, der entweder zentral oder verteilt zu führen ist, bestimmt zu werden, womit dann die jeweils richtigen Objektversionen ausgewählt werden können. Eine weniger aufwendige Alternative ergibt sich, wenn sich die benötigten Versionen aus der Rechner-ID und der lokalen Uhrzeit ableiten lassen [38, 43].

5.3 Konsistenzebene 2

In kommerziellen DBS, in denen zur Zeit durchweg Sperrverfahren zur Synchronisation eingesetzt werden, wird aus Leistungsgründen meist auf Serialisierbarkeit verzichtet, sondern lediglich die sogenannte Konsistenzebene 2 [52] eingesetzt. Das bedeutet, daß nur Schreibsperren bis zum TA-Ende, Lesesperren dagegen kurz (im allgemeinen für die Dauer der jeweiligen DB-Operation) gehalten werden, wodurch natürlich weniger Konflikte als mit langen Lesesperren verursacht werden. Eine Anomalie, die dabei in Kauf genommen wird, ist „unrepeatable read" [53]; d.h., eine TA kann von demselben Datenobjekt unterschiedliche, jedoch gültige Versionen sehen. Dies wird meist als akzeptabel angesehen, zumal viele TA ein Objekt ohnehin nur einmal referenzieren.

Eine Beschränkung auf Konsistenzebene 2 läßt sich auch bei optimistischer Synchronisation zu einer erheblichen Reduzierung der Konflikthäufigkeit nutzen; außerdem ergibt sich eine deutliche Verringerung des Validierungsaufwandes. Um Konsistenzebene 2 zu gewährleisten, gilt es nämlich nur die Anomalien „dirty read", „dirty overwrite" und „lost update" [53] zu vermeiden. Dabei sind jedoch die ersten beiden Anomalien schon dadurch umgangen, weil alle Änderungen in einem privaten TA-Puffer vorbereitet werden und die Schreibphasen, durch Blockieren der zu schreibenden Objekte für die Dauer der Schreibphase, atomar gehalten werden können. Zur Behebung von „lost update" brauchen aber – sowohl bei FOCC als auch bei BOCC bzw. BOCC+ – nur noch Änderungs-TA gegenüber anderen Änderungs-TA validiert zu werden [27]. Lese-TA jedoch sind – wie beim Multiversion-Konzept – weder selbst zu validieren, noch brauchen sie bei der Validierung berücksichtigt zu werden. Damit erhält man dieselben Vorteile (erhebliche Reduzierung von Rücksetzungen und Validierungsaufwand sowie Kommunikationseinsparungen in VDBS) wie mit einem Mehrversionen-Ansatz, aber ohne den Aufwand der Versionenverwaltung; außerdem ergeben sich für Mehrrechner-DBS keine zusätzlichen Schwierigkeiten.

Gegenüber einem (RX-)Sperrverfahren mit langen Schreib- und kurzen Lesesperren bietet die skizzierte Vorgehensweise zwei wesentliche Vorteile. Zum einen werden mit der Validierung von Änderungs-TA gegenüber anderen Änderern „lost updates" definitiv vermieden, während bei Sperrverfahren mit kurzen Lesesperren Änderungen noch verlorengehen können, falls die Lesesperren zu kurz gehalten werden [27]. Der zweite Vorteil besteht darin, daß Lese-TA bei optimistischer Synchronisation weitaus weniger behindert werden als bei dem Sperrverfahren, weil die einzigen synchronisationsbedingten Verzögerungen für Leser das kurzzeitige Warten auf das Ende einer Schreibphase sind; Änderungs-TA warten zudem nie auf Lese-TA. Beim RX-Sperrverfahren dagegen können für Leser durch die langen Schreibsperren erhebliche Wartezeiten entstehen. Die höhere Parallelität bei optimistischer Synchronisation wird ermöglicht, da Änderungen in einer privaten Kopie vorbereitet werden, so daß Lesern (außer während dem Einbringen einer Änderung in der Schreibphase) stets eine konsistente Objektversion angeboten werden kann. Dieser Vorteil läßt sich demnach auch prinzipiell mit Sperrverfahren erreichen, bei denen die Änderungen auf privaten Kopien vorgenommen werden und Leser trotz solcher Änderungen auf die ungeänderten Objektversionen zugreifen können (z.B. RAX-Sperrverfahren mit kurzen Lesesperren).

Andererseits kann mit kurzen Lesesperren für die Dauer der Sperre eine Wiederholbarkeit von Lesevorgängen garantiert werden, was für einige Anwendungen von Bedeutung ist („cursor stability"). In einigen DBS wird es sogar durch spezielle DML-Befehle (z.B. KEEP/FREE bei UDS) ermöglicht, Lesesperren über die Dauer der aktuellen DB-Operation hinweg zu halten. Eine solcherart erreichbare, zeitweise Wiederholbarkeit von Lesevorgängen läßt sich aber auch bei optimistischer Synchronisation in analoger Weise herstellen, indem nun Lese-TA auch für die jeweilige Zeitdauer „Sperren" setzen. Diese Sperren müssen nun von validierenden Änderungs-TA beachtet werden, wodurch die Änderung gesperrter Objekte verhindert wird und für die Sperrdauer Wieder-

holbarkeit von Lesevorgängen und „cursor stability" gesichert ist.

Dieses kurzzeitige Sperren gelesener Objekte führt für die Lese-TA weder zu zusätzlichen Verzögerungen noch zu Deadlocks; lediglich für Änderungs-TA besteht eine etwas erhöhte Rücksetzgefahr. In VDBS führt das Sperren zwar bei zentraler Validierung zu zusätzlichen Kommunikationen, diese fallen aber auch bei einem rein pessimistischen Verfahren mit zentralem Lock-Manager an.

6. Resümee und Ausblick

In dieser Arbeit wurden ein Überblick über die wichtigsten Vorschläge für optimistische Synchronisationsverfahren in zentralisierten und verteilten Datenbanksystemen gegeben sowie einige neue Techniken vorgestellt. Wie bisherige Leistungsanalysen bestätigt haben, ist das ursprüngliche BOCC-Verfahren nach [2] wegen seiner vielen Schwachpunkte (2.1) kein ernstzunehmender Konkurrent für Sperrverfahren. Wesentlich leistungsfähiger sind dagegen das BOCC+- und das FOCC-Verfahren, bei denen Rücksetzungen wegen unechter Konflikte und das Aufbewahren von Write-Sets beendeter TA umgangen werden sowie ein vorzeitiges Zurücksetzen von TA (zum Einsparen unnötiger Arbeit) und parallele Schreibphasen möglich sind. FOCC hat noch den Vorteil einer flexiblen Konfliktauflösung, während bei BOCC+ die schnellsten Validierungen möglich sein dürften (sehr wenig Vergleiche).

Die beiden optimierten Verfahren sind ausreichend für Anwendungen mit geringer Konflikthäufigkeit, für die optimistische Verfahren zunächst auch ausschließlich vorgesehen wurden. Das Einsatzspektrum läßt sich jedoch erweitern, wenn eine der drei in Kap. 5 angesprochenen Techniken zur Reduzierung von Rücksetzungen angewendet wird. So ist Lese-TA ihr Durchkommen bei einem Multiversion-Konzept bzw. bei Konsistenzebene 2 immer gesichert; es ergeben sich wesentlich weniger Rücksetzungen sowie ein verringerter Validierungsaufwand, da nur noch Änderungs-TA validieren bzw. bei der Validierung berücksichtigt werden müssen. Für Lese-TA ergeben sich so in VDBS auch Kommunikationseinsparungen, da sie nicht zu validieren brauchen. Ein Multiversion-Konzept ist dabei weniger interessant, da es bei Sperrverfahren analog einsetzbar ist und einen relativ hohen Verwaltungsaufwand und Speicherplatzbedarf einführt. Eine andere Optimierungsmöglichkeit ist, daß eine validierende Lese-TA möglicherweise in der Serialisierungsreihenfolge vor bereits validierte TA eingereiht werden kann, wenn sie zwar einen leicht veralteten, jedoch konsistenten DB-Zustand gesehen hat. Eine elegante Realisierungsform dazu wird durch die Verwendung von Zeitintervallen möglich, wie in 5.1 beschrieben.

Verglichen mit Sperrverfahren haben diese optimistischen Verfahren den Vorteil der Deadlock-Freiheit sowie einer potentiell höheren Parallelität bzw. besseren Antwortzeiten wegen der i. a. geringeren Anzahl von Blockierungen. Letzteres wird ermöglicht durch die Änderungen in einer privaten Kopie, die für Leser stets eine konsistente Objektversion erreichbar läßt. Auch hinsichtlich der algorithmischen Komplexität bzw. des Laufzeit-Overheads können sich Nachteile für Sperrverfahren ergeben, da verglichen mit den zum Teil sehr einfachen Validierungen (z. B. bei BOCC+) Sperrkonversionen oder die Erkennung von Deadlocks sehr aufwendig sind.

Wenn bei optimistischer Synchronisation die gleiche Universalität wie für Sperrverfahren angestrebt wird, dann reichen in konfliktträchtigen Anwendungen (lange Änderungs-TA, änderungsintensive Hot Spots) selbst die stark verbesserten Verfahren nicht mehr aus, da dann zu viele Rücksetzungen sowie möglicherweise zyklische Restarts zu befürchten sind. In diesen Fällen ist daher eine Kombination zwischen optimistischer und pessimistischer Synchronisation vorzunehmen. So ist eine pessimistische Synchronisation z. B. für lange Änderungs-TA und bereits gescheiterte TA angebracht; bei Konsistenzebene 2 läßt sich durch das Setzen von Lesesperren auch Wiederholbarkeit von Lesevorgängen („cursor stability") erreichen.

Bei optimistischen Synchronisationsverfahren in verteilten Datenbanksystemen können die Validierungen entweder zentral oder verteilt erfolgen. Eine zentrale Validierung erlaubt eine relativ einfache Realisierung und verursacht weniger Kommunikationsaufwand (eine synchrone Nachricht pro TA; bei den Optimierungen nach Kap. 5 nur eine pro Änderungs-TA) als bei einem zentralen Sperrverfahren. Dennoch erscheint der zentrale Ansatz weniger empfehlenswert, weil auch für rein lokale TA Kommunikation zur Validierung notwendig wird und der zentrale Knoten eine besondere Ausfallbehandlung verlangt. Außerdem kann nur eine BOCC-artige Validierung eingesetzt werden, so daß vor allem zur Vermeidung zyklischer Restarts eine Kombination mit Sperrverfahren unabdingbar erscheint. Dabei sind jedoch für pessimistisch synchronisierte TA viele globale Sperranforderungen zu erwarten; der Preclaiming-Ansatz ist primär auf kurze TA beschränkt.

Die verteilte Validierung findet in Kombination mit einem Zweiphasen-Commit-Protokoll statt, um eine eindeutige Validierungsentscheidung treffen und die Atomizität der TA sicherstellen zu können. Dieses zweiphasige Validierungsschema ist jedoch nur für globale TA erforderlich, da für lokale TA die lokale Validierung ausreicht. Ein weiterer Vorteil liegt darin, daß sowohl eine BOCC- als auch eine FOCC-Strategie anwendbar ist; bei Kombination mit einem Sperrverfahren wird (außer zur Deadlock-Erkennung) kein zusätzlicher Kommunikationsaufwand eingeführt. Für

eine korrekte Synchronisation ist eine Koordinierung der verschiedenen Validierungen einer TA sowie eine geeignete Behandlung von (beabsichtigten) Änderungen von „semicommitted" TA erforderlich. Die einfachste Vorgehensweise wird hierbei möglich, wenn auf jedem Knoten die Validierungen zu den TA in der gleichen Reihenfolge vorgenommen werden (kann z. B. mit Broadcast- oder Token-Ring-Ansatz erreicht werden) und der Zugriff auf zu ändernde Objekte blockiert wird, bis der Ausgang der Änderungs-TA feststeht.

Der Kommunikationsaufwand eines rein optimistischen Verfahrens mit verteilter Validierung entspricht in etwa dem eines verteilten Sperrverfahrens ohne Deadlock-Erkennung. Wird beim Sperrverfahren zur Behandlung von Deadlocks nur ein einfaches Timeout-Verfahren bzw. eine Deadlock-Vermeidung praktiziert, dann ist bei ihnen die Anzahl der Rücksetzungen nicht notwendigerweise geringer als für optimistische Verfahren, insbesondere wenn eine der Optimierungen für Lese-TA angewendet wird. Eine explizite Deadlock-Erkennung erhöht zwar den Kommunikationsoverhead, scheint aber vor allem in konflikthäufigeren Anwendungen sinnvoll. Quantitative Leistungsvergleiche zwischen optimistischen Verfahren und Sperrverfahren in verteilten Umgebungen liegen jedoch bisher noch kaum vor (in [54] schnitt der optimistische Ansatz besser ab, jedoch führen die Autoren das zum Teil auf E/A-Engpässe zurück, da für das Sperrverfahren vor jeder Änderung ein Before-Image geschrieben wurde).

Die quantitativen Leistungsuntersuchungen beschränkten sich bisher vorwiegend auf das ursprüngliche BOCC-Verfahren in zentralisierten DBS. Um den optimistischen Verfahren gerecht zu werden, gilt es nun vor allem realitätsgetreue Leistungsanalysen der verbesserten Protokolle vorzunehmen, sowohl für zentralisierte DBS als für verteilte Umgebungen. Damit die Konflikthäufigkeit nicht unnötig erhöht wird, ist weiterhin bei der Implementierung optimistischer Protokolle auf ein möglichst kleines Synchronisationsgranulat zu achten (Satz, Eintrag); weitere Implementierungsfragen betreffen die Systempufferverwaltung, Logging und Recovery. Auf verteilte DBS mit partieller oder voller Redundanz konnte in dieser Arbeit ebenfalls nicht eingegangen werden; hier sind die Verfahren analog zu den für Sperrverfahren vielfach angegebenen Lösungsmöglichkeiten zur Konsistenzerhaltung der Kopien [55, 56] zu erweitern. Schließlich sollte auch für optimistische Verfahren das Ausmaß der Synchronisationskonflikte durch Nutzung der (möglicherweise anwendungsbezogenen) Zugriffssemantik auf Objekte (z. B. Kommutativität bestimmter Operationen) reduziert werden können. Hiermit können z. B. gerade auf High-Traffic-Objekten viele Konflikte vermieden werden [57].

Danksagung

Für wertvolle Hinweise zu einer vorläufigen Fassung dieser Arbeit bin ich Herrn Prof. Dr. T. Härder sowie Herrn T. Wagner zu Dank verpflichtet.

Literatur

[1] *Eswaran, K. P.; Gray, J. N.; Lorie, R. A.; Traiger, I. L.:* The Notions of Consistency and Predicate Locks in a Database System. CACM 19 (11), 624–633 (1976).

[2] *Kung, H. T.; Robinson, J. T.:* On Optimistic Methods for Concurrency Control. ACM TODS 6 (2), 213–226 (1981).

[3] *Roome, W. D.:* The Intelligent Store: A Content-Adressable Page Manager. Bell System Tech. Journal 61 (9), 2567–2596 (1982).

[4] *Fishman, D. H.; Lai, M.; Wilkinson, W. K.:* Overview of the JASMIN Database Machine. Proc. ACM SIGMOD, 234–239 (1984).

[5] *Kersten, M. L.; Tebra, H.:* Application of an Optimistic Concurrency Control Method. Software – Practice and Experience 14 (2), 153–168 (1984).

[6] *Leland, M. D. P.; Roome, W. D.:* The Silicon Database Machine. Proc. 4th Int. Workshop on Database Machines, 169–189, Springer (1985).

[7] *Mullender, S. J.; Tanenbaum, A. S.:* A Distributed File Service Based on Optimistic Concurrency Control. Proc. 10th ACM Symp. on Operating System Principles, 51–62 (1985).

[8] *Härder, T.:* Observations on Optimistic Concurrency Control. Information Systems 9 (2), 111–120 (1984).

[9] *Unland, R.:* Optimistische Synchronisationsverfahren und ihre Leistungsfähigkeit im Vergleich zu Sperrverfahren. Dissertation, Fernuniversität Hagen (1985).

[10] *Menasce, D. A.; Nakanishi, T.:* Optimistic versus Pessimistic Concurrency Control Mechanisms in Database Management Systems. Information Systems 7 (1), 13–27 (1982).

[11] *Morris, R. J. T.; Wong, W. S.:* Performance of Concurrency Control Algorithms with Nonexclusive Access. In: Performance 84, pp. 87–101, North-Holland 1984.

[12] *Morris, R. J. T.; Wong, W. S.:* Performance Analysis of Locking and Optimistic Concurrency Control Algorithms. Performance Evaluation 5 (2), 105–118 (1985).

[13] *Agrawal, R.; DeWitt, D. J.:* Integrated Concurrency Control and Recovery Mechanisms: Design and Performance Evaluation. ACM TODS 10 (4), 529–564 (1985).

[14] *Carey, M. J.; Stonebraker, M. R.:* The Performance of Concurrency Control Algorithms for Database Management Systems. Proc. 10th Int. Conf. on VLDB, 107–118 (1984).

[15] *Augustin, R.; Scholten, H. A.; Prädel, U.:* Modelling Database Concurrency Control Algorithms using a General Purpose Performance Evaluation Tool. In: Performance 84, pp. 69–85, North-Holland 1984.

[16] *Agrawal, R.; Carey, M. J.; Liny, M.:* Models for Studying Concurrency Control Performance: Alternatives and Implications. Proc. ACM SIGMOD, 108–121 (1985).

[17] *Peinl, P.:* Synchronisation in zentralisierten Datenbanksystemen – Algorithmen, Realisierungsmöglichkeiten und quantitative Bewertung –. Dissertation, Fachbereich Informatik, Univ. Kaiserslautern (1986).

[18] *Schlageter, G.:* Optimistic Methods for Concurrency Control in Distributed Database Systems. Proc. 7th Int. Conf. on VLDB, 125–130 (1981).

[19] *Unland, R.; Prädel, U.; Schlageter, G.:* Design Alternatives for Optimistic Concurrency Control Schemes. Proc. 2nd Int. Conf. on Databases (ICOD-2), 288–297 (1983).

[20] *Prädel, U.; Schlageter, G.; Unland, R.:* Einige Verbesserungen optimistischer Synchronisationsverfahren. Proc. 12. GI-Jahrestagung, Informatik-Fachberichte 57, 684–698, Springer 1982.

[21] *Thomasian, A.; Ryu, I. K.:* Analysis of Some Optimistic Concurrency Control Schemes Based on Certification. Proc. SIGMETRICS, 192–203 (1985).

[22] *Gold, I.; Shmueli, O.; Hofri, M.:* The Private Workspace Model Feasibility and Applications to 2PL Performance Improvements. Proc. 11th Int. Conf. on VLDB, 192–208 (1985).

[23] *Lausen, G.:* Concurrency Control in Database Systems: A Step Towards the Integration of Optimistic Methods and Locking. Proc. ACM Annual Conf., 64–68 (1982).

[24] *Boral, H.; Gold, I.:* Towards a Self-Adapting Centralized Concurrency Control Algorithm. Proc. SIGMOD, 18–32 (1984).

[25] *Vidyasankar, K.; Raghavan, V. V.:* Highly Flexible Integration of the Locking and the Optimistic Approaches of Concurrency Control. Proc. IEEE COMPSAC, 489–494 (1985).

[26] *Gawlick, D.:* Processing „Hot Spots" in High Performance Systems, Proc. IEEE Spring CompCon, 249–251 (1985).

[27] *Rahm, E.:* Optimistische Synchronisationsverfahren in Datenbanksystemen: Ein Überblick. Interner Bericht 166/87, Fachbereich Informatik, Univ. Kaiserlautern (1987).

[28] *Bayer, R.; Elhardt, K.; Kießling, W.; Killar, D.:* Verteilte Datenbanksysteme – Eine Übersicht über den heutigen Entwicklungsstand. Informatik Spektrum 7 (1), 1–19 (1984).

[29] *Härder, T.; Rahm, E.:* Mehrrechner-Datenbanksysteme für Transaktionssysteme hoher Leistungsfähigkeit. Informationstechnik 28 (4), 214–225 (1986).

[30] *Härder, T.; Rahm, E.:* Hochleistungs-Datenbanksysteme – Vergleich und Bewertung aktueller Architekturen und ihrer Implementierung. Informationstechnik 29 (3), 127–140 (1987).

[31] *Badal, D. Z.:* Correctness of Concurrency Control and Implications in Distributed Databases. Proc. COMPSAC, 588–593 (1979).

[32] *Bhargava, B.:* Resiliency Features of the Optimistic Concurrency Control Approach for Distributed Database Systems. Proc. 2nd Symp. on Reliability in Distributed Software and Database Systems, 19–32 (1982).

[33] *Badal, D. Z.; McElyea, W.:* A Robust Adaptive Concurrency Control for Distributed Databases. Proc. IEEE INFOCOM84, 382–391 (1984).

[34] *Bernstein, P. A.; Goodman, N.:* Concurrency Control in Distributed Database Systems. ACM Computing Surveys 13 (2), 185–221 (1981).

[35] *Rahm, E.:* Concurrency Control in DB-Sharing Systems. Proc. 16. GI-Jahrestagung, Informatik-Fachberichte 126, 617–633, Springer 1986.

[36] *Reuter, A.; Shoens, K.:* Synchronization in a Data Sharing Environment. Vorläufige Fassung, IBM San Jose Research Lab (1984).

[37] *Ceri, S.; Owicki, S.:* On the Use of Optimistic Methods for Concurrency Control in Distributed Databases. Proc. 6th Berkeley Workshop on Distributed Data Management and Computer Networks, 117–129 (1982).

[38] *Lai, M.; Wilkinson, K.:* Distributed Transaction Management in JASMIN. Proc. 10th Int. Conf. on VLDB, 466–470 (1984).

[39] *Ghertal, F. F.; Mamrat, S.:* An Optimistic Concurrency Control Mechanism for an Object Based Distributed System. Proc. 5th Int. Conf. on Distr. Comp. Systems, 236–245 (1985).

[40] *Sinha, M. K.; Nanadikar, P. D.; Mehndiratta, S. L.:* Timestamp, Based Certification Schemes for Transactions in Distributed Database Systems. Proc. SIGMOD, 402–411 (1985).

[41] *Sheth, A. P.; Liu, M. T.:* Integrating Locking and Optimistic Concurrency Control in Distributed Database Systems. Proc. 6th Int. Conf. on Distr. Comp. Systems, 89–99 (1986).

[42] *Boksenbaum, C.; Cart, M.; Ferrie, J.; Pons, J.-F.;* Concurrent Certifications by Intervals of Timestamps in Distributed Database Systems. IEEE Trans. on Software Engineering 13 (4), 409–419 (1987).

[43] *Agrawal, D.; Bernstein, A. J.; Gupta, P.; Sengupta, S.:* Distributed Optimitic Concurrency Control with Reduced Rollback. Distributed Computing 2 (1), 45–59 (1987).

[44] *Schlageter, G.:* Problems of Optimistic Concurrency Control in Distributed Database Systems. ACM SIGMOD Record 12 (3), 62–66 (1982).

[45] *Härder, T.; Peinl, P.; Reuter, A.:* Optimistic Concurrency Control in a Shared Database Environment. Manuskript, Univ. Kaiserslautern/Stuttgart (1985).

[46] *Prädel, U.; Schlageter, G.; Unland, R.:* Redisgn of Optimistic Methods: Improving Performance and Applicability. Proc. 2nd Data Engineering Conf., 466–473 (1986).

[47] *Bayer, R.; Elhardt, K.; Heigert, J.; Reiser, A.:* Dynamic Timestamp Allocation for Transactions in Database Systems. Proc. 2nd Int. Symp. Distributed Data Bases, 9–20 (1982).

[48] *Chan, A.; Fox, S.; Lin, W.; Nori, A.; Ries, D.:* The Implementation of an Integrated Concurrency Control and Recovery Schema. Proc. ACM SIGMOD, 184–191 (1982).

[49] *Carey, M. J.:* Multiple Versions and the Performance of Optimistic Concurrency Control. Technical Report No. 517, Comp. Science Department, Univ. of Wisconsin-Madison (1983).

[50] *Carey, M. J.; Muhanna, W. A.:* The Performance of Multiversion Concurrency Control Algorithms. ACM Trans. on Comp. Systems 4 (4), 338–378 (1986).

[51] *Chan, A.; Dayal, U.; Hsu, M.:* Providing Database Management Capabilities for Mission Critical Applications. Proc. Int. Workshop on High Performance Transaction Systems, Asilomar (1985).

[52] *Gray, J. N.; Lorie, R. A.; Putzolu, G. R.; Traiger, I.:* Granularity of Locks and Degrees of Consistency in a Shared Data Base. Proc. IFIP Working Conf. on Modelling in Data Base Management Systems, 365–394, North-Holland (1976).

[53] *Gray, J.:* Notes on Data Base Operating Systems. In: Bayer, R. Graham, R. M., Seegmüller, G. (eds.): Operating Systems – an advanced course. Lecture notes in Computer Science, Vol. 60, pp. 393–481. Berlin, Heidelberg, New York, Tokyo: Springer 1978.

[54] *Kohler, W. J.; Jenq, B. P.:* Performance Evaluation of Integrated Concurrency Control and Recovery Algorithms using a Distributed Transaction Processing Testbed. Proc. 6th Int. Conf. on Distr. Computing Systems, 130–139 (1986).

[55] *Davidson, S. B.; Garcia-Molina, H.; Skeen, D.:* Consistency in Partitioned Networks. ACM Comp. Surveys 17 (3), 341–370 (1985).

[56] *Son, S. H.:* Synchronization of Replicated Data in Distributed Systems. Information Systems 12 (2), 191–202 (1987).

[57] *Reuter, A.:* Concurrency on High-Traffic Data Elements. Proc. PODS, 83–93 (1982).

Diese Arbeit wurde von der SIEMENS AG finanziell unterstützt.