Prof. Dr. Peter Dadam

Kurs 01666

Datenbanken in Rechnernetzen

LESEPROBE

Fakultät für
Mathematik und Informatik

FernUniversität in Hagen

Leseprobe zu 1666: Datenbanken in Rechnernetzen

8. Replikationsverfahren

8.3 Motivation

Ein verteiltes System ist, bedingt durch die größere Anzahl von Komponenten, bei gleicher Güte der Komponenten, inhärent anfälliger gegen Ausfälle als ein zentrales System. Werden stets alle Komponenten des verteilten Systems benötigt, dann addieren sich im Wesentlichen die Ausfallwahrscheinlichkeiten. Bei einer entsprechend großen Anzahl von Komponenten ist das Gesamtsystem dann praktisch so gut wie nicht mehr einsatzfähig.

Diese Gesetzmäßigkeit gilt natürlich auch für verteilte Informationssysteme. Der Ausweg heißt hier (und auch allgemein bei verteilten Systemen) gezielte Verwendung von redundanten Komponenten, um im Fehlerfall, möglichst transparent für die Anwendungen, die Verarbeitung mit der „Ersatzkomponente“ durchführen zu können. Bei verteilten Informationssystemen ist in der Regel eine *hohe Verfügbarkeit der Daten das* zentrale Anliegen. Deshalb heißt hier die Lösung *redundante Speicherung von Daten*; nicht notwendigerweise aller Daten, aber doch zumindest der bzgl. Verfügbarkeit sehr kritischen Daten. Wie wir im Folgenden noch sehen werden, reicht allerdings eine s*imple Mehrfachspeicherung* der kritischen Daten, insbesondere in Verbindung mit einer *simplen Updatestrategie* nicht aus, um die Verfügbarkeit eines verteilten Informationssystems zu erhöhen. Im Gegenteil, man kann hierdurch sogar leicht den gegenteiligen Effekt erzielen. Der Wahl eines geeigneten, für die Anwendungen passenden *Replikationsverfahrens* kommt in diesem Zusammenhang daher eine große Bedeutung zu.

Eine andere potentielle Schwachstelle verteilter (Informations-)Systeme ist der zu treibende *Kommunikationsaufwand*, der additiv zu den lokal entstehenden Zugriffs- und Verarbeitungskosten hinzukommt. Bei besonders zeitkritischen Anwendungen wird man daher bestrebt sein, den Kommunikationsaufwand dadurch zu reduzieren oder sogar ganz zu vermeiden, dass man die Daten (Partitionen) so allokiert, dass möglichst häufig ein *lokaler Zugriff* möglich ist und dadurch die Antwortzeit verkürzt wird (siehe Kapitel 3, insbesondere Abschnitt 3.8). Wird der Zugriff von mehreren Knoten auf dieselben Daten benötigt und ist dieser Zugriff (zumindest von einigen dieser Knoten) überwiegend lesend, so wird man eine *redundante Speicherung* (Allokation) dieser Daten in Erwägung ziehen. Damit stellt sich dann aber ebenfalls das *Problem der Aktualisierung bzw. Konsistenthaltung* dieser redundant gespeicherten Daten. Auch hier ist die Wahl des richtigen *Replikationsverfahrens* sehr entscheidend dafür, ob der beabsichtigte Performanzgewinn auch tatsächlich realisiert werden kann.

Im nächsten Abschnitt wollen wir zunächst einmal die grundsätzlichen Problemstellungen und Vorgehensweisen näher betrachten, bevor wir dann im darauffolgenden Abschnitt auf einige Verfahren konkreter eingehen.

8.4 Grundsätzliche Problemstellungen und Vorgehensweisen

In den folgenden Unterabschnitten wollen wir die verschiedenen Problemstellungen und Lösungsansätze zunächst allgemein betrachten. In Abschnitt 9.5 werden wir dann ausgewählte, für eine bestimmte Richtung wegweisende bzw. charakteristische Verfahren kennenlernen. Mit einer Ausnahme, auf die wir dann explizit hinweisen werden, wird im Folgenden der Begriff *Kopie* als Synonym für *redundante Allokation* eines Datenelementes (Tupel, Relation / Partition, ...) verwendet. Es gibt also kein speziell ausgewiesenes „Original", sondern die „Kopien" sind gewissermaßen alle (replizierte) Originale.

8.4.1 Read-One-Write-All-Verfahren (ROWA-Verfahren)

Im Idealfall sollte sich für die Anwendungen bzw. Benutzer eine eventuelle Replikation der Daten nur durch eine Verbesserung der Antwortzeit (lokaler Zugriff) bemerkbar machen. Im Fehlerfall sollte das System im Prinzip eine beliebige andere *Kopie* der benötigten Daten zur Weiterarbeit nutzen können. Dies impliziert natürlich, dass alle Kopien stets auf dem gleichen Stand sind.

Wie man sich leicht überlegt, lässt sich das am einfachsten dadurch erreichen, dass bei Änderungen stets *alle Kopien* der betroffenen Daten innerhalb derselben Update-Transaktion geändert werden. Dies ist auch implementierungstechnisch einfach: man braucht nur die Update-Anweisung entsprechend der Anzahl der Kopien zu vervielfachen. Bei dieser Vorgehensweise sind stets *alle Kopien auf demselben Stand*. Da beim Update alle Kopien gleichzeitig gesperrt sind, kann auch keine Lesetransaktion „versehentlich" einen veralteten oder gar inkonsistenten Zustand lesen. Eine Lesetransaktion (und in diesem Fall natürlich auch eine Updatetransaktion) kann sich also im Prinzip eine beliebige Kopie auswählen (man bezeichnet dieses Verfahren daher auch als *Read-One-Write-All-Verfahren* (*ROWA-Verfahren*)). Die Dauer einer Updateanweisung, insbesondere die Durchführung des 2PC-Protokolls (siehe Abschnitt 7.6), wird hierbei durch die „langsamste" Kopie bestimmt.

Für Lesetransaktionen ist damit der ROWA-Ansatz ideal. Leider gilt dies *nicht für Updatetransaktionen*. Ist nur eine der Kopien nicht erreichbar, sind keine Updates auf dem betroffenen Datenbestand mehr möglich. (Nicht alle Subtransaktionen erreichen ihren RtC-Zustand[1], der globale Update schlägt damit fehl.) Die Verfügbarkeit des Systems für Updatetransaktionen sinkt also mit jeder (zusätzlichen) Kopie. Damit ist das *ROWA-Verfahren in der reinen Form* für den praktischen Einsatz in der Regel *nicht geeignet*.

In den letzten Jahren wurde eine sehr große Anzahl von Replikationsverfahren entwickelt (siehe /BeDa95/ für einen Überblick) und jedes Jahr kommen neue Vorschläge hinzu. Wir wollen im Folgenden schrittweise eine gewisse Kategorisierung für die verschiedenen Ansätze entwickeln, um die Einordnung der verschiedenen Verfahren zu erleichtern.

Bei der Diskussion von Replikationsverfahren kann man im Wesentlichen vier Punkte unterscheiden:

---

[1] siehe Abschnitt 7.6 in Kurseinheit 5

1. Kopien-Update-Strategie:
   Vorgehensweise im Normalfall, d. h. Anzahl und Auswahl der Kopien, die zur Durchführung eines Updates benötigt werden.

2. Fehlerbehandlung:
   Vorgehensweise im Fehlerfall, insbesondere bei Netzpartitionierungen.

3. Synchronisation konkurrierender Zugriffe:
   Eingesetztes Synchronisationsverfahren und Gewährleistung der globalen Serialisierbarkeit von Transaktionen.

4. Behandlung von Lesetransaktionen.

Auf diese Punkte wollen wir in den folgenden Abschnitten nun nacheinander eingehen.

8.4.2 Kopien-Update-Strategien

Bis auf das in Abschnitt 9.4.1 besprochene ROWA-Verfahren gehen alle anderen Verfahren davon aus, dass für die Durchführung eines Updates eines Datenelementes nur eine Teilmenge der hierzu existierenden Kopien benötigt wird. Die Verfahren unterscheiden sich in der Anzahl der benötigten Kopien und in der Art ihrer Bestimmung. Abb. 9-1 zeigt die verschiedenen Verfahren im Überblick und gibt eine Einordnung derjenigen Verfahren, die wir in Abschnitt 9.5 näher behandeln werden.

8.4.2.1 Verfahren mit vorbestimmter Kopie

Bei den Verfahren dieser Kategorie, deren bekanntester (und ältester) Vertreter das *Primary-Copy-Verfahren* /Ston79/ ist, wird eine bestimmte Kopie als *Primärkopie* oder *Masterkopie* festgelegt. Sie ist die *Originalversion*, alle anderen sind von dieser abgeleitete Kopien. Ein Update erfolgt bei diesem Verfahren zweistufig: Die Updatetransaktionen sperren und ändern im Wesentlichen nur die Primärkopie. Die Aktualisierung der anderen Kopien übernimmt dann die Primärkopie in eigener Regie, nach Commit der jeweiligen Updatetransaktion. Für Updatetransaktionen gestaltet sich die Durchführung von Updates damit relativ einfach.

Etwas problematischer gestaltet sich das konsistente und aktuelle lokale Lesen sowie die Fehlerbehandlung bei Ausfall der Primärkopie. Wir werden hierauf in Abschnitt 9.5.1 noch näher eingehen.

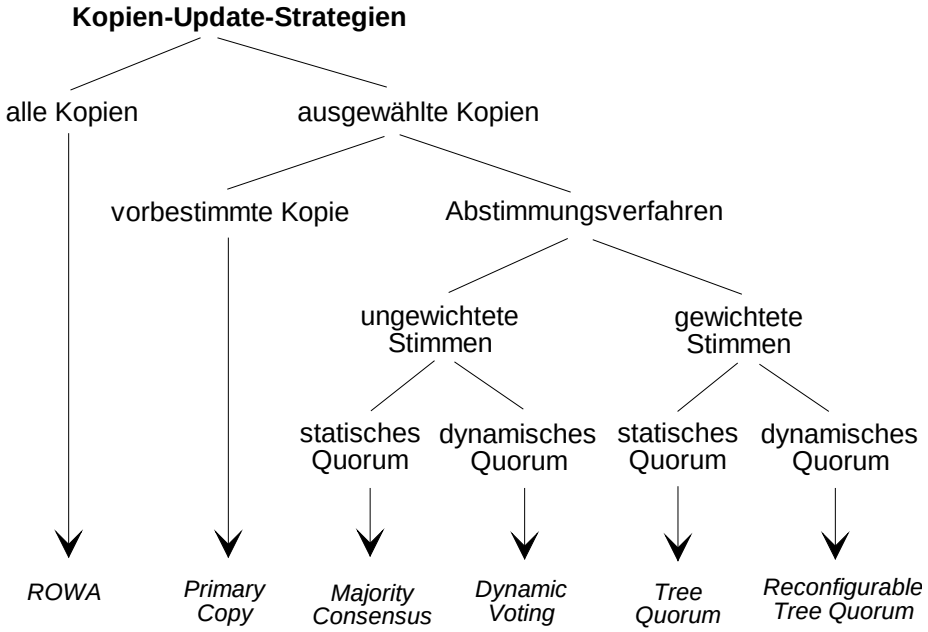

Abb. 9-1: Kopien-Update-Strategien im Überblick

8.4.2.2 Abstimmungsverfahren

Bei den Verfahren dieser Kategorie wird ein Update auf einer Kopie dann durchgeführt, wenn die entsprechende Transaktion in der Lage ist, eine Mehrheit von Kopien hierfür zu gewinnen (also z.B. geeignet zu sperren). Die Verfahren unterscheiden sich zum einen darin, ob alle Kopien bzgl. dieser Abstimmung gleich behandelt werden (*ungewichtete Stimmen*) oder ob den Kopien in gewisser Weise unterschiedliche Stimmgewichte zugeordnet werden (*gewichtete Stimmen*), und zum anderen darin, ob die jeweilige Anzahl von Stimmen zum Erreichen einer Mehrheit fest vorgegeben ist (*statisches Quorum*) oder ob dies erst zur Laufzeit bestimmt wird (*dynamisches Quorum*).

Statisches vs. dynamisches Quorum

Bei den *quorumbasierten Ansätzen* wird über einen Lesezugriff bzw. über einen Update in gewisser Weise durch die beteiligten Kopien „abgestimmt“. Erreicht ein entsprechender „Antrag“ eine genügend große Anzahl von Stimmen, so ist er angenommen und der Zugriff wird durchgeführt, ansonsten wird er abgelehnt. Eine „genügend große“ Anzahl von Stimmen kann im einfachsten Fall bedeuten, dass jeweils mehr als die Hälfte der betroffenen Kopien zustimmen muss (also $n \div 2 + 1$ bei n Kopien[2]). Es ist jedoch auch eine andere Quorumseinteilung denkbar. Um etwa Lesetransaktionen gegenüber Updatetransaktionen zu bevorzugen, könnte man festlegen, dass eine Lesetransaktion lediglich 1/3 der Stimmen benötigt, während eine Updatetransaktion mehr als 2/3 der Stimmen auf sich vereinigen muss. Um Serialisierbarkeit zu gewährleisten, muss das Quorum für Lesetransaktionen und Updatetransaktionen so gewählt werden, dass weder eine Lese- und eine Updatetransaktion noch zwei Updatetransaktionen gleichzeitig die Zustimmung erhalten können.

---

[2] Das Zeichen ÷ steht für *ganzzahlige Division*. Es gilt also z. B. $7 \div 2 = 3$.

Etwas formaler ausgedrückt:

Sei Q das Gesamtquorum (also die Gesamtzahl der erreichbaren Stimmen), sei Q_L das für einen Lesezugriff benötigte Quorum und sei Q_U das für einen Update benötigte Quorum, dann müssen Q_L und Q_U so gewählt werden, dass stets gilt:

1. $Q_L + Q_U > Q$ (Behandlung Lese-Schreib-Konflikte)
2. $Q_{U_1} + Q_{U_2} > Q$ (Behandlung Schreib-Schreib-Konflikte)

Beim *statischen Quorum* wird die Anzahl der benötigten Stimmen von der Anzahl der beim Systemstart vorhandenen Kopien dieses Datenelementes abgeleitet. Sind also z. B. 10 Kopien bei Systemstart vorhanden und werden $n \div 2 + 1$ Zustimmungen (also mind. 6 Stimmen) für einen Zugriff benötigt, so kann bei Ausfall von 5 Knoten keine Mehrheit für einen Zugriff mehr erreicht werden.

Beim *dynamischen Quorum* versucht man dieses Problem dadurch zu vermeiden, dass man das Lese-/Schreibquorum dynamisch an die *Anzahl der aktuell verfügbaren Knoten* anpasst. Dies gewährleistet in der Regel bei Knotenausfällen eine höhere Verfügbarkeit des Gesamtsystems, verursacht allerdings auch im Normalbetrieb einen erhöhten Buchführungs- und Koordinationsaufwand. Wir werden hierauf in den Abschnitten 9.5.3 und 9.5.5 noch eingehen.

Ungewichtete vs. gewichtete Stimmen

Bei der ersten Kategorie (*ungewichtete Stimmen*) werden alle Kopien gleich behandelt. D. h. sie werden bei Bedarf im Prinzip in beliebiger Reihenfolge angefragt und um Stimmabgabe gebeten. Bei einem Updatequorum Q_U werden somit mindestens Q_U Kopien angefragt. Um die Anzahl der Nachrichten zur Erlangung des benötigten Quorums zu reduzieren, wurde eine Reihe von Verfahren vorgeschlagen. Diese versuchen, durch eine strukturierte Vorgehensweise bei der Abstimmung bzw. durch die Einführung *verschiedener Stimmengewichte*, die Anzahl der im Mittel anzufragenden Kopien möglichst klein zu halten, so dass also weniger als $n \div 2 + 1$ Kopien im Updatefall angefragt werden müssen.

Ein sehr einfaches Beispiel eines solchen gewichteten Quorums wäre z. B. bei 7 Kopien einer „Hauptkopie“ 5 Stimmen und den restlichen 6 Kopien je eine Stimme zuzuordnen. Das Gesamtquorum beträgt damit 11 Stimmen. Die benötigte Mehrheit von 6 Stimmen kann damit entweder mit der Hauptkopie plus einer weiteren Stimme oder mit allen 6 Einzelstimmen erreicht werden. In Abschnitt 9.5.4 werden wir ein Verfahren kennenlernen, das Hierarchie von Stimmgewichten einsetzt und damit zu einer baumartigen Abstimmungsstruktur kommt.

8.4.3 Strategien für den Fehlerfall

Das ROWA-Verfahren ausgenommen, haben alle anderen Verfahren Maßnahmen zur „Fehlertoleranz“ integriert. Bei der Diskussion der vorgesehenen Konzepte bietet es sich an, zwischen dem Verhalten bei *Ausfall einzelner Kopien* und dem Verhalten bei *Netzpartitionierung* (das Rechnernetz zerfällt durch Kommunikationsunterbrechungen temporär in isolierte Teilnetze) zu unterscheiden.

8.4.3.1 Ausfall einzelner Knoten

Vorbestimmte Kopie

Wird beim Update mit einer vorbestimmten Kopie wie beim Primary-Copy-Verfahren gearbeitet, so ist diese Hauptkopie dafür zuständig, ausgefallene Kopien beim Wiederanlauf mit der aktuellen Version zu versorgen. Fällt die Primärkopie selbst aus, so ist es im Prinzip möglich, eine der anderen Kopien zur Primärkopie zu „küren“, wobei unter Umständen allerdings ein gewisser Aktualitätsverlust, je nach Update-Propagations-Strategie der Primärkopie, in Kauf genommen werden muss. Bei der *Erzeugung einer neuen Primärkopie* muss sichergestellt sein, dass die Primärkopie tatsächlich ausgefallen ist und nicht etwa nur durch eine Netzpartitionierung für einige Knoten unerreichbar geworden ist.[3,4]

Abstimmungsverfahren

Bei den Abstimmungsverfahren führt - je nach Verfahren - ohnehin nur ein Teil der Knoten den Update synchron durch, die anderen Knoten werden ggf. asynchron aktualisiert. Der Ausfall einzelner Knoten ist deshalb hinsichtlich der Behandlung der verbliebenen Knoten relativ ähnlich zum Normalfall (verzögerte Aktualisierung), nur dass in diesem Fall die Updateinformation länger aufbewahrt werden muss.

8.4.3.2 Netzpartitionierung

Hinsichtlich des Verhaltens bei Netzpartitionierung kann man die in Abb. 9-2 dargestellten Fälle unterscheiden. Die Einordnung der Verfahren, die wir in Abschnitt 9.5 besprechen werden, ist dort ebenfalls wieder angegeben. Bezüglich des Verhaltens bei Netzpartitionierung kann man zwei Gruppen von Verfahren unterscheiden: *konservative* und *progressive* Verfahren.

---

[3] Dies ist ein Beispiel dafür, warum ein verteiltes DBMS in gewissem Umfang Einblick in und Kontrolle über das zugrundeliegende Kommunikationssystem haben sollte. Wir hatten dies bereits in Abschnitt 2.7 in Kurseinheit 1 erwähnt.

[4] Wir werden auf diesen Punkt bei der Behandlung des Primary-Copy-Verfahrens in Abschnitt 9.5.1 nochmals zu sprechen kommen.

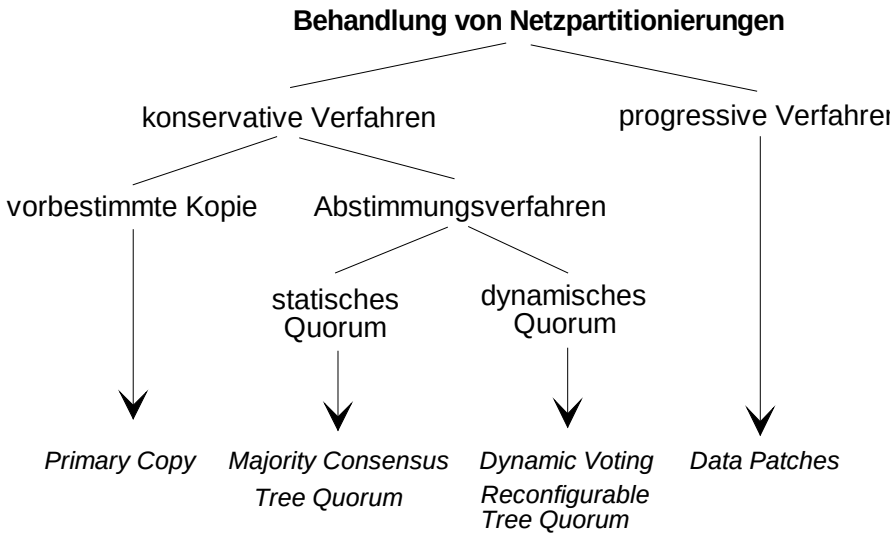

Abb. 9-2: Verhalten bei Netzpartitionierung

Konservative Verfahren

In der Gruppe der konservativen Verfahren, der die meisten Verfahren zuzurechnen sind, hat die Erhaltung der *Konsistenz* der Datenbank die *höchste Priorität*. Hierfür nimmt man ggf. auch Performanzeinbußen oder auch die temporäre Nichtverfügbarkeit des Systems in Kauf. Typisch für diese Gruppe von Verfahren ist, dass im Falle einer Netzpartitionierung, bei der Kopien desselben Datenelements in getrennten Teilnetzen liegen, nur in einem Teilnetz - im Folgenden *Hauptpartition* genannt - Updates auf diesem Datenelement durchgeführt werden dürfen. Die Unterschiede zwischen den einzelnen Verfahren bestehen vor allem darin, wie die Hauptpartition bestimmt wird und bis zu welchem Grad an Netzpartitionierung die Hauptpartition noch bestimmbar bzw. arbeitsfähig ist.

Wird bzgl. Update-Koordination mit einer *vorbestimmten Kopie* gearbeitet, wie etwa beim Primary-Copy-Verfahren, so bestimmt die *Zugehörigkeit der Primärkopie* im Falle einer Netzpartitionierung die Hauptpartition. Das Teilnetz, das die Primärkopie enthält, ist uneingeschränkt arbeitsfähig, während die anderen Teilnetze allenfalls noch lesenden Zugriff auf die vorhandenen Kopien gestatten.

Bei den *Abstimmungsverfahren* wird die Hauptpartition über die Mehrheit der Kopien definiert. Das Teilnetz, das über die Mehrheit der Kopien eines Datenelements verfügt, wird bezüglich dieses Datenelements zur Hauptpartition. Verfahren mit statischer und dynamischer Quorumbildung unterscheiden sich im Wesentlichen in der Behandlung mehrfacher Netzpartitionierungen. Während bei den statischen Verfahren an dem einmal vorgegebenen Quorum festgehalten wird, wird dies bei den dynamischen Verfahren bei Netzpartitionierung ggf. verkleinert und bei der Behebung der Störung ggf. wieder vergrößert. Wir werden hierauf bei der Behandlung des Dynamic-Voting-Verfahrens in Abschnitt 9.5.3 nochmals zu sprechen kommen.

Progressive Verfahren

Bei den progressiven Verfahren wird der *Verfügbarkeit* des Gesamtsystems die *höchste Priorität* eingeräumt. Bei Auftreten einer Netzpartitionierung darf in allen Teilnetzen

uneingeschränkt weitergearbeitet, also auch Updates durchgeführt werden. Hierfür nimmt man ggf. temporäre Inkonsistenzen der Datenbank in Kauf und vertraut darauf, dass diese bei der Wiedervereinigung erkannt und beseitigt werden können. Ein interessanter Vertreter dieser Kategorie ist das Verfahren *Data Patches* /Garc83/, auf das wir in Abschnitt 9.5.6 noch näher eingehen werden.

8.4.4 Synchronisation von Updatetransaktionen

In diesem Abschnitt werden wir uns vor allem mit der Bewahrung der Konsistenz des verteilten Informationssystems bzw. der verteilten Datenbank im Kontext von *Updates* befassen. Mit Lesetransaktionen werden wir uns gezielt in Abschnitt 9.4.5 auseinandersetzen.

8.4.4.1 Korrektheitsaspekte

Die Verwendung von Replikaten soll, wie bereits eingangs dieses Kapitels erwähnt, lediglich zur Verbesserung der Systemverfügbarkeit und/oder zur Verbesserung des Durchsatzes für lesende Zugriffe dienen. Ansonsten soll sich das (verteilte) System „nach außen“ wie ein (verteiltes) System ohne Kopien verhalten. Man sagt deshalb auch, dass das Verfahren aus Sicht der Anwendung *1-Kopie-äquivalent* (1-copy equivalent) sein soll und erweitert dazu den Serialisierbarkeitsbegriff entsprechend zur *1-Kopie-Serialisierbarkeit*:

Definition 9-1: 1-Kopie-Serialisierbarkeit

Eine Schedule S von (abgeschlossenen) Transaktionen, die auf einer repliziert gespeicherten verteilten Datenbank ausgeführt wurden, heißt dann und nur dann *1-Kopie-serialisierbar*, wenn es mindestens eine serielle Ausführung der Transaktionen aus S auf einer verteilten Datenbank ohne Replikate gibt, welche, angewandt auf denselben Ausgangszustand, die gleiche Ausgabe sowie denselben Endzustand erzeugt.

Definition 9-1 besagt letztlich, dass in einem verteilten System mit Kopien dieselben Korrektheitsprinzipien wie in einem System ohne Kopien gelten. Mit entsprechenden Synchronisationsverfahren, die dies beachten, wollen wir uns nun in den nächsten Abschnitten befassen.