Replikation verteilter Datenbanken

Frank Büchel

Fakultät Informatik, Mathematik und Naturwissenschaften
HTWK Leipzig

10. Juli 2015

1 Motivation

2 Verfahren

3 Primary Copy

4 Abstimmungsverfahren

5 Ausfalltolleranz

6 Konfliktlösung

Motivation

- Kritische Datenbestände von Unternehmen und Einrichtungen
  - Redundantes Ersatzsystem bei Ausfall
- Dezentral agierende Unternehmen
  - Verkürzung der Antwortzeiten durch lokale Zugriffe
- Addition der Ausfallwahrscheinlichkeiten
- $\hookrightarrow$ Anfälliger für Ausfall durch Komponentenanzahl

Anforderungen

- Möglichst transparente Weiterverarbeitung durch Ersatzkomponente
- Verhalten wie zentrales System
- Anforderung: hohe Verfügbarkeit der Daten
  - Verteilte, redundante Speicherung kritischer Daten
  - Wahl eines Anwendungsbezogenen Replikationsverfahrens

Schwachstellen

- Überwiegender Lesezugriff von Knoten auf selbe Daten
  $\hookrightarrow$ redundante Speicherung
- Allokation für häufige Zugriffe auf lokale Knoten
- Aktualisierung aller Änderungen
- Probleme:
  - Erhöhter Kommunikationsaufwand
  - Additiv zu entstehenden Verarbeitungs- und Zugriffskosten
  - Dauer der Aktualisierung vs. Konsistenz
- Bei Zeitkritischen Anwendungen Kommunikationsaufwand reduzieren oder vermeiden

Motivation

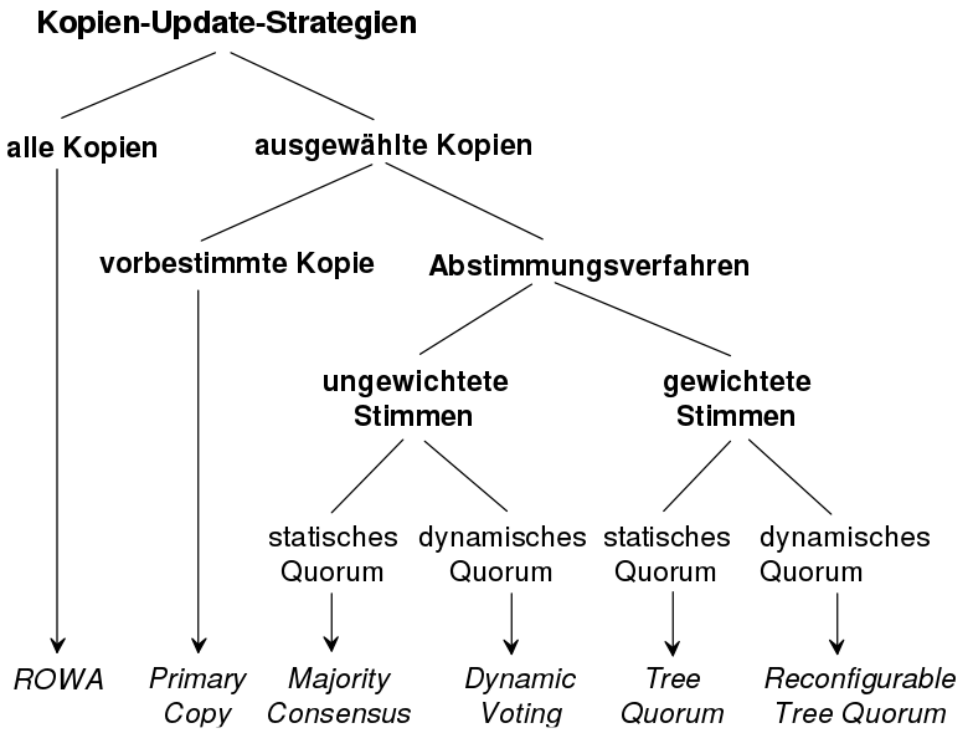


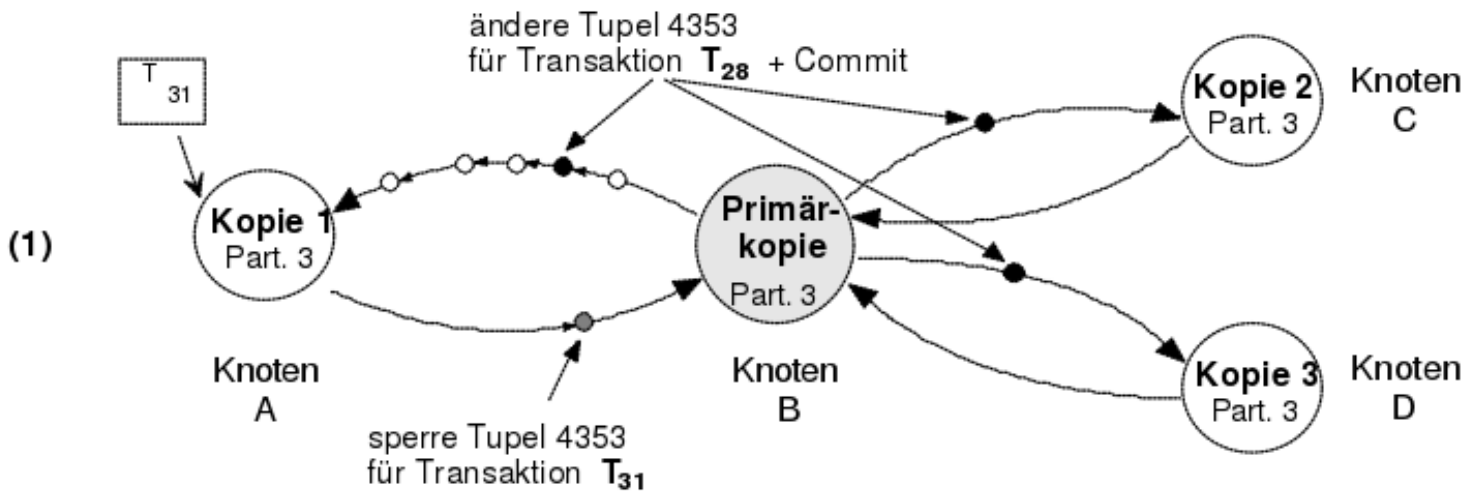
Abbildung: Strategien zur Replikation verteilter Datenbanken[1]

Primary-Master-Replikation

- Bestimmung einer Kopie als Primär- bzw. Masterkopie
- Entspricht der Originalversion, alle Anderen abgeleitet
- Zweistufiges Updateverfahren:
  - U-TRX sperren und ändern nur Master
  - Asynchrone Aktualisierung übriger Kopien durch Master
  - Replikation nach COMMIT von UTRX
  - Einfache Durchführung für Updatetransaktionen

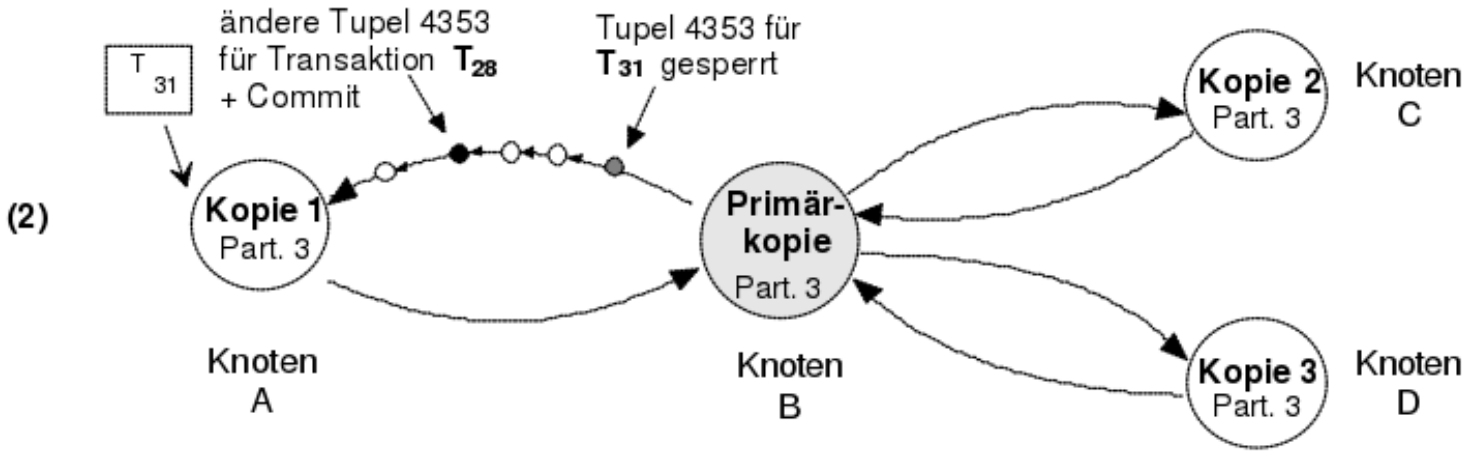
Funktionsweise (1)

- Start T_{31} an Knoten A $\rightarrow$ ändern von Tupel 4353
- Sperranforderung über Sendekanal an Primärkopie
- Gleichzeitig Update-Auftrag an Empfangskanal Knoten A
- Betrifft Änderung an Tupel 4353 durch T_{28}



Funktionsweise (2)

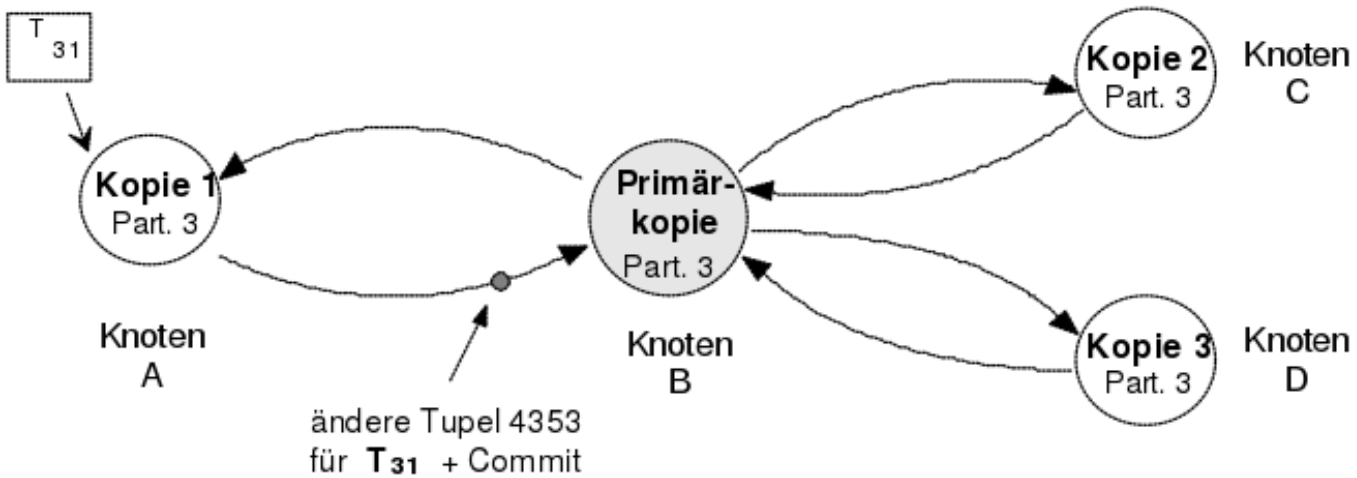
- Knoten A wartet auf Antwort der Sperranforderung
- Dabei Abarbeitung der Nachrichten und Änderungsaufträge im Epfangskanal



Funktionsweise (3)

- Knoten A fertig mit Abarbeitung inklusive Bestätigung der Sperrgewährung
- Lokale Kopie von Tupel 4353 gleich zu Primärkopie
- Übertragung des geänderten Wertes
- Aufforderung zum Commit

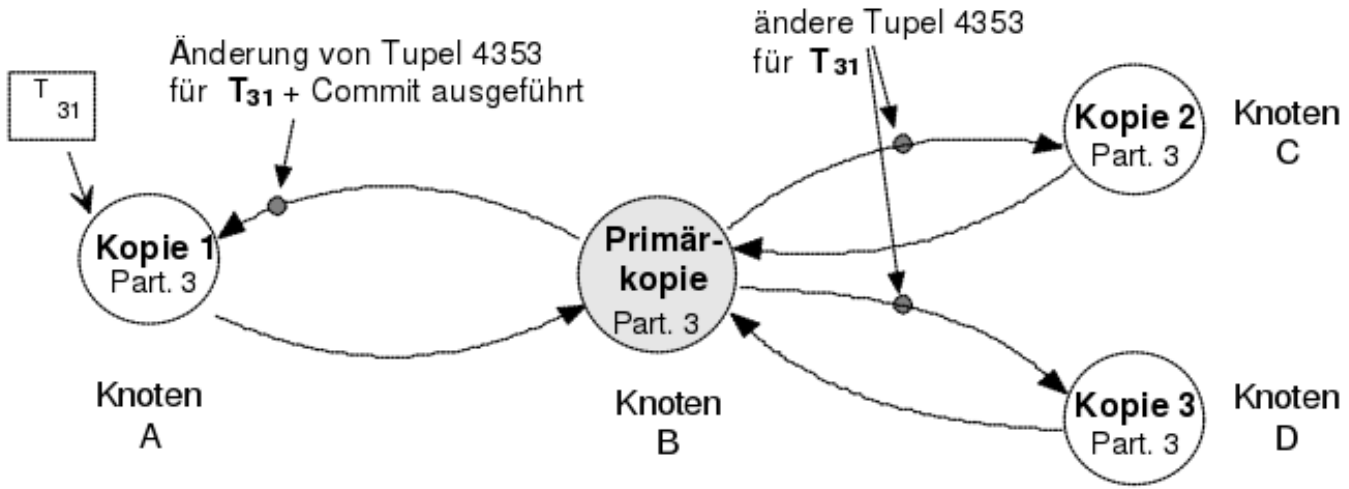
(3)



Funktionsweise (4)

- Bestätigung des Updates durch Commit von Primärkopie
- Master startet Aktualisierung übriger Kopien

(4)



Abstimmungsverfahren

- Durchführung eines Update nur wenn Mehrheit der Kopien Update zustimmen (z.B. Sperren)
- Mehrheit (51%) nicht erreicht $\rightarrow$ TRX abgelehnt
- Beteiligte Knoten stimmen über Lesezugriffe und Updates ab
- Andere Einteilung zur Bevorzugung von L-TRX denkbar
- Lesen $Q_L = \frac{1}{3}$, Ändern $Q_U = \frac{2}{3}$

- Serialisierbarkeit: gleichzeitige Zustimmung L-TRX und U-TRX oder U-TRX und U-TRX vermeiden

Formal:

- $Q_L + Q_U > Q$ (Behandlung Lese- Schreibkonflikte)
- $Q_U + Q_U > Q$ (Behandlung Schreib- Schreib Konflikte)

statische vs. dynamische Quoren

Statisches Quorum

- Festlegen der Quorenanzahl zur Startzeit
- $N =$ Anzahl der Knoten
- 10 Knoten $\rightarrow Q = \frac{N}{2} + 1 = 6$ Stimmen
- Bei Ausfall von 5 Knoten ist Mehrheit nicht erreichbar

Dynamisches Quorum

- Dynamisches Anpassen an aktuell verfügbare Knoten
- Führt bei Knotenausfällen zu höherer Verfügbarkeit des Gesamtsystems
- Problem: höhere Buchhführungskosten und Koordinationsaufwand

ungewichtete vs. gewichtete Stimmen

Ungewichtete Stimmen

- Gleichberechtigte Anfrage zur Stimmanbgabe an alle Kopien
- Anfrage in beliebiger Reihenfolge
- Mindestens Q_U Kopien werden angefragt

Gewichtete Stimmen

- Strategien zur Reduktion des Nachrichtenaufwandes
- Also weniger als n/2+1 Anfragen

Beispiel: gewichtete Stimmen

- $N = 7$ Kopien
- Hauptkopie: 5 Stimmen
- 6 Kopien je 1 Stimme
- $\hookrightarrow Q = 11$
- Mehrheit, wenn Hauptkopie + 1 Kopie oder 6 Einzelstimmen

Ausfalltolleranz

- Alle Verfahren mit ausgewählten Kopien sind Fehlertollerant
- Ausfall einzelner Knoten
- Ausfall durch Netzpartition
  - Kommunikationsunterbrechung
  - Zerfall des Netzes in Teilnetze (Allokationen)

Primary-Copy-Verfahren:

- Master ist Zuständig für Kopie-Aktualisierung bei Wiederanlauf
- Bei Ausfall des Masters kann neue Primärkopie gewählt werden
- Aktualitätsverlust möglich
- Zu Erzeugung eines neuen Masters muss Ausfall der Vorherigen sichergestellt sein

Abstimmungsverfahren

- Synchronisierung ohnehin partiell auf Teilmenge der Knoten
- Rest asynchron aktualisiert
- Ausfall ähnlich zu Normalfall
- Aktualisierung lediglich verzögert
- Aufbewahrung der Updateinformationen über längeren Zeitraum

Konservative Verfahren

- Einhaltung der Konsistenz hat Priorität (CP)
- Inkaufnahme von Performanceeinbußen und Nichtverfügbarkeit
- Kopien des selben Datenbestandes in getrennten Teilnetzen
- Updates nur in einem Teilnetz (Hauptpartition)

Bestimmung der Hauptpartition

Primary-Copy

- Allokation des Masters zu Hauptpartition bestimmt
- Hauptpartition uneingeschränkt funktionsfähig
- Rest der Partitionen gewähren nur Lesezugriff

Abstimmungsverfahren

- Bestimmung der Hauptpartition über Mehrheit der Kopien
- Allokation mit Mehrheit bei Update wird Hauptpartition
- Fall dynamische Quorenbildung: Verkleinerung des Quorums
- Problem: Inkonsistenzen bei Wiedervereinigung

Progressive Verfahren

- Verfügbarkeit hat höchste Priorität (AP)
- Uneingeschränkte Transaktionsverarbeitung in allen Allokationen
- Inkaufnahme temporärer Inkonsistenzen
- Konflikterkennung und Konfliktlösung bei Wiedervereinigung

Konfliktlösung

- Lösungsregeln bereits zum Entwurf festgelegt
- Festlegung für jede kritische Relation
- Unterscheidung: Einfügung und Aktualisierung
- Festlegung von Einfüge- und Integrationsregeln

Regeln (1)

Annahme Netzzerfall in A_1 und A_2
Tupel in A_1 eingefügt

Einfügeregeln

- `KEEP`: Tupel einfügen in A_2
- `REMOVE`: Tupel löschen in A_1
- `PROGRAM`: Aufruf eines Programms mit Tupel als Eingabeparameter
- `NOTIFY`: Meldung an Administrator zur manuellen Konfliktlösung

Regeln (2)

Tupel in A_1 *und/oder* in A_2 geändert

Integrationsregeln

- `LATEST`: zuletzt eingebrachte Änderung gilt
- `PRIMARY k`: Tupel an Knoten k gilt
- `ARITHMETIK`: neuer Wert = Wert1 + Wert2 - alter Wert
- `PROGRAM`
- `NOTIFY`

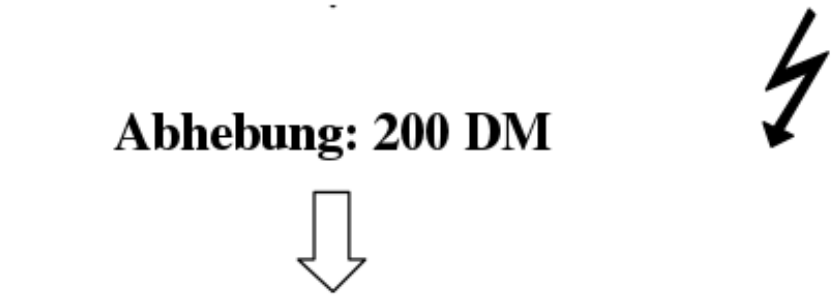
Konto-Relation

Konto#	Name	Kontostand	Einfügungsregel	Integrationsregel
			KEEP	ARITHMETIC

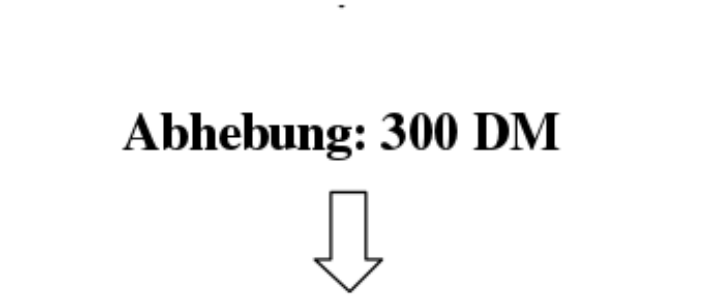
Ausgangssituation:

Knoten 1		
Konto#	Name	Kontostand
1723	Maier	1.000 DM

Knoten 2		
Konto#	Name	Kontostand
1723	Maier	1.000 DM



Knoten 1		
Konto#	Name	Kontostand
1723	Maier	800 DM



Knoten 2		
Konto#	Name	Kontostand
1723	Maier	700 DM

Wiedervereinigung: Integrationsregel = ARITHMETIC

$$\begin{aligned} \Rightarrow \text{Kontostand}_{neu} &:= \text{Kontostand}_1 + \text{Kontostand}_2 - \text{Kontostand}_{alt} \\ &= 800\ \text{DM} + 700\ \text{DM} - 1.000\ \text{DM} \\ &= 500\ \text{DM} \end{aligned}$$

Nach Konfliktauflösung:

Knoten 1		
Konto#	Name	Kontostand
1723	Maier	500 DM

Knoten 2		
Konto#	Name	Kontostand
1723	Maier	500 DM

Danke für das Interesse!

Quelle:

Peter Dadam. *Verteilte Datenbanken und Client/Server-Systeme - Grundlagen, Konzepte und Realisierungsformen.* Springer, 1996, S. I–XII, 1–415. ISBN: 978-3-540-61399-2.