

# Verteilte Systeme

## 14. Transaktionen

### Motivation

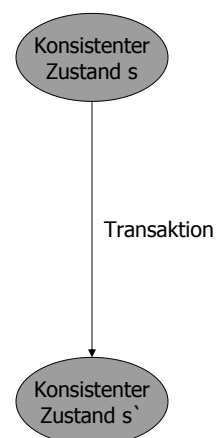
#### Sicherung konsistenter Systemzustände

#### Beispiele

- Amnesieproblematik bei zustandsbehafteten Servern
- Sicherung des Primaries (Primary-Backup-Approach)
- Aktive Fehlertoleranz?
- Concurrency Control

#### Transaktionen

- Von konsistenten Zuständen in konsistente Zustände überführen
- Auch im Fehlerfall
- Auch in Konkurrenz zu parallel ausgeführten Transaktionen



### Beispiel: Konsistentes Logging

Realisierung At-Most-Once-Semantik

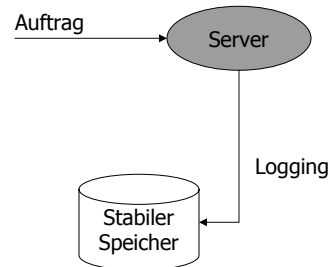
- Sicherung der Replies

Primary-Backup-Approach

- Datenzustände

Anforderungen

- Sicherung der Zustandsänderungen
- Wiederherstellbarkeit von Zuständen
- Wiederholen von Aufträgen
- Undo von angebrochenen Aufträgen
- „Alles oder nichts“



Verteilte Systeme, Sommersemester 1999

Folie 14.3

### Beispiel: Concurrency Control

Inkonsistenzen durch zeitlich verschränkte Zugriffe

Lösungsmöglichkeiten

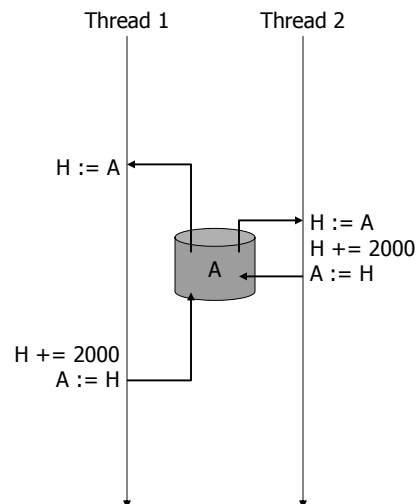
- Kritische Abschnitte

Anwendungsspezifische Definition atomarer Aktionen

- **BAA** ;
- ...
- EAA** ;

Transaktionen

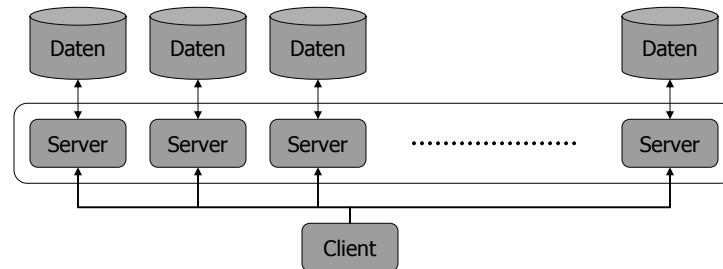
- Konsistenz gemeinsam benutzter Variablen



Verteilte Systeme, Sommersemester 1999

Folie 14.4

### Beispiel: Replikation von Zuständen



Dateisysteme, Namensdienst, Ensemble  
 Konsistente Aktualisierung aller Replikate  
 Serverabstürze während der Aktualisierung

Verteilte Systeme, Sommersemester 1999

Folie 14.5

### Transaktionen



#### **A**tomic

- Jede Transaktion ist unteilbar (alles oder nichts)

#### **C**onsistency

- Jede Transaktion überführt einen konsistenten Zustand in einen konsistenten Zustand

#### **I**solation (Serialisierbarkeit)

- Die konkurrente Ausführung mehrerer Transaktionen ist identisch zu einer sequentiellen Ausführungsreihenfolge

#### **D**urability

- Erfolgreich beendete Transaktionen überleben nachfolgende Fehler

Verteilte Systeme, Sommersemester 1999

Folie 14.6

## Bemerkungen

Anwendungsunabhängig realisierbar

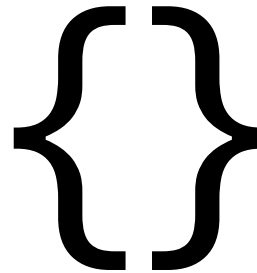
- Atomarität
- Isolation
- Dauerhaftigkeit

Konsistenz ist anwendungsspezifisch

Klammerung

- Beginn einer Transaktion: BOT
- Ende einer Transaktion: EOT
- Erfolgreich: Commit
- Nicht erfolgreich: Abort

Einmal Commit, immer Commit?



## Realisierungsvarianten

Zeitpunkt der Sichtbarkeit von Zustandsänderungen

Pessimistische Systeme

- Sichtbarkeit, wenn erfolgreich beendet und gesichert
- Einschränkung der Parallelität (Sperrern)

Optimistische Systeme

- Alle Änderungen sofort sichtbar
- Nachträgliche Überprüfung auf Konflikte bei EOT und ggf. Zurücknahme der Änderungen
- Höhere Parallelität (Keine Sperrern)
- Gefahr sogenannter Abort-Kaskaden

## 14.1 Pessimistische Systeme

### 2-Phasen-Sperren

#### Gängiges Verfahren

Daten, auf die eine Transaktion zugreift, werden gesperrt

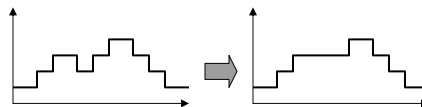
- Andere Transaktionen können nicht darauf zugreifen
- Änderungen erst nach dem Commit sichtbar (Aufheben der Sperren)

#### Unterschiedliche Sperrgranulate

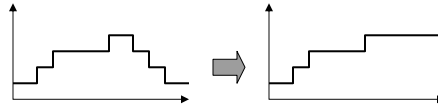
- Lesen und Schreiben
- Vgl. Reader/Writer
- Verstärkung und Abschwächung von Sperren

#### 2 Phasen

- Anforderungsphase
- Freigabephase



## Striktes 2-Phasen-Sperren



Freigabe aller Sperren erst bei EOT

- Sperren länger als notwendig (Einschränkung der Parallelität)

Warum?

Deadlockgefahr

- Ausschließen (Prevention)
  - Sperren aufsteigend anfordern
  - Sperren länger als notwendig
- Auflösen (Detection)
  - Erkennen verteilter Deadlocks
  - Aufbrechen des Deadlocks (Ist das schwer?)

## 14.2 Optimistisch Systeme

# 14.3 Recovery

## Recovery

---

Wiederherstellung des konsistenten Zustands nach Fehler

3 Speicherklassen

- Flüchtiger Hauptspeicher
- Nicht-flüchtige Platten und Bänder
- Stabiler Speicher

Recovery-Stufen

- Abbruch einer Transaktion
- Systemabsturz
- Plattenfehler
- Katastrophe

The diagram illustrates the memory hierarchy and recovery components. At the top, a box contains the CPU and volatile memory (Flüchtiger Speicher). Below this, a line connects to a cylinder representing non-volatile memory (Nichtflüchtiger Speicher). To the left, a separate box shows a door with a keyhole and a key, with a cylinder representing stable storage (Stabiler Speicher) inside. This suggests that stable storage is a secure, persistent location for data recovery.

Verteilte Systeme, Sommersemester 1999 Folie 14.14

## Abbruch einer Transaktion

### Update-in-place

#### Update

- Speicherung eines UNDO-Satzes
- Aktualisierung der Originaldaten

#### Read

- Lesen der Originaldaten

#### Commit

- UNDO-Sätze löschen

#### Abort

- Änderungen mit Hilfe der UNDO-Sätze rückgängig machen

### Deferred-Write

#### Update

- Änderungen in REDO-Liste speichern

#### Read

- Letzter Wert (REDO-Liste rückwärts durchsuchen bis Originaldaten)

#### Commit

- Änderungen mittels REDO-Liste nachziehen

#### Abort

- REDO-Liste löschen

## Systemabsturz

Updates (UNDO- und REDO-Listen) sowie Commit oder Abort werden auf einem sequentiellen Logfile im nicht-flüchtigen Speicher gesichert

### Redo-Rule

- Commit muß im Logfile stehen (physisch), bevor Transaktion als erfolgreich beendet gilt

### Undo-Rule (Write-ahead log rule)

- UNDO-Satz muß im Logfile stehen (physisch), bevor Originaldaten geändert werden dürfen

### Beschränkung der Logfile-Größe

- Invalidierung von UNDO- und REDO-Einträgen
- Checkpointing



## Plattenfehler

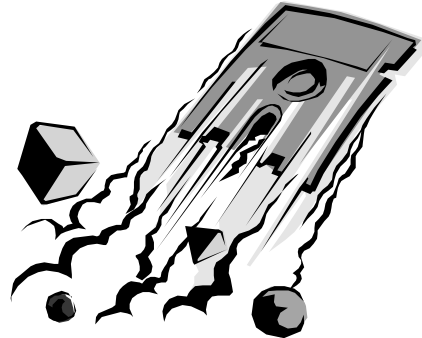
Archivkopie der Daten einschließlich einem Archiv-Log

Archivkopie n-fach replizieren

- Größe von n?

Konsistentes Archiv

- System einfrieren
- Nachteil: Lange Auszeiten
- Fuzzy Dumps



Verteilte Systeme, Sommersemester 1999

Folie 14.17

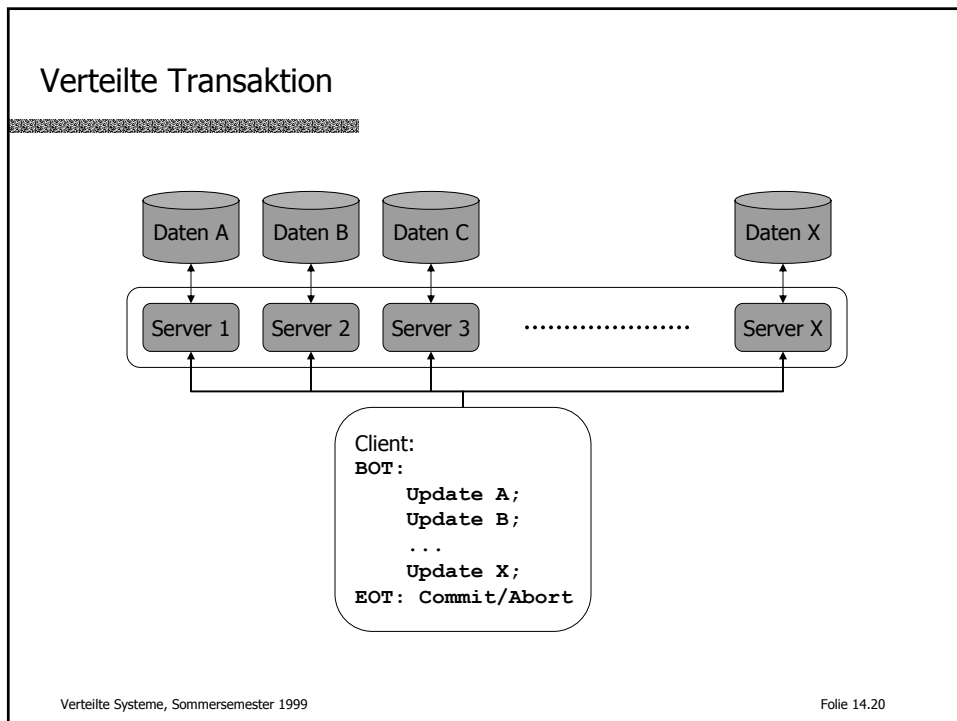
## Katastrophe



Verteilte Systeme, Sommersemester 1999

Folie 14.18

# 14.4 Verteilte Transaktionen



## Commit-Protokolle

---

Jeder Rechner kann lokal für Commit und Abort entscheiden

Eine Entscheidung ist nach dem Publizieren nicht mehr änderbar

### Agreement

- Alle Rechner kommen zur gleichen Entscheidung
- Commit, wenn alle Rechner sich für Commit entscheiden
- Abort, wenn mindestens ein Rechner für Abort entscheidet

### Commit-Protokolle

- Agreement auch im Fehlerfall herbeiführen

## 2-Phasen-Commit

---

### Koordinator

- Z.B. Knoten, auf dem Transaktion begonnen wurde

### Phase 1

- Koordinator sendet Commit-Request an alle
- Teilnehmer antworten mit Ja oder Nein

### Phase 2

- Koordinator sammelt Antworten
- Entscheidung
  - Commit: Alle haben mit Ja geantwortet
  - Abort: Mindestens einer hat mit Nein geantwortet
- Ergebnis allen Teilnehmern mitteilen

## Fehlerfälle

Commit-Request kommt nicht an

- Teilnehmer warten angemessene Zeit (Timeout) und entscheiden sich bei Ausbleiben für Abort

Teilnehmerantwort kommt nicht an

- Koordinator entscheidet nach Timeout für Abort

Ergebnis des Koordinators kommt nicht an

- Lokale Entscheidung Nein: Okay
- Lokale Entscheidung Ja: Gossip
- Blockadesituation !

## Übungsaufgaben

1. Versuchen Sie, ein nicht-blockierendes Commit-Protokoll zu definieren.