Transaktionale Informationssysteme SoSe19

Benedikt Lüken-Winkels

July 2, 2019

Contents

1 Vorlesung

- **Vorlesung** Di, 14:15-15:45, H11
- **Übung** Mo, 13-14
- **Prüfung** mündlich 16.06. und 22.10.

Motivation

Bei vielen, kurzen Transaktionen (Änderungen) darf die Datenbasis nicht zerstört werden

- Rollback
- Administration der Aktionen auf der Datenbasis
- $\Rightarrow$ Datenkonsistenz

Konsistenz Bewahrung der Korrektheit Daten im Fehlerfall

Generizität Abstraktion von Szenarien

Paralleler Zugriff Beispiel 1.1 (Folie 12)

Naive Parallelverarbeitung sorgt zum Konflikt

Optimistischer Ansatz Laufen lassen, bis ein Fehler Auftritt

Pessimistische Ansatz Zugriff blockieren

Fehlerhafte Ausführung Beispiel 1.2 (Folie 13)

Prozess wird durch Fehler unterbrochen

Rollback Sollten nicht alle Aktionen ausführbar sein, nicht ausführen (Komplett oder gar nicht)

Verteiltes Datensystem Beispiel 1.3 (Folie 14)

Verschiedene Datenbestände nicht korrekt synchronisiert (zB Client- und Serverwarenkorb), Datensysteme sind verschieden und unahängig voneinander (heterogen und autonom)

Transaktionale Eigenschaften

- Synchronisierung von Client und Serverinformationen
- Verifikation des Abschlusses einer Transaktion

Beispiel 1.4 (Folie 19)

Gesamte Aktion muss erfolgreich sein: Schlägt eine Transaktion im Block fehl, wirf eine Fehlermeldung (zB Prüfungsanmeldung und Bestätigung)

Workflow Management

Spezifikation von Workflows

- Wer bekommt welche Rolle

Workflow

- Geschäftsprozess (zB Beschaffung, Reiseplanung)
- Langlebig

Aktivität Teile eines Workflows, die von verschiedenen Akteuren augeführt werden

Architekturen

Einfache Server Struktur (Folie 27) Data Server: Datendatendarstellung

- Gekapselt in Objekten (Request, Reply)
- Ungekapselt als Tupel

Föderierte Systeme

- Alte Systeme müssen mit neuen Systemen kooperieren

Transaktionsmanagement

ACID (Folie 30)

- Atomarität: Ganz oder gar nicht
- Consistenz: Konsistenzerhaltung, waren die Daten Konsistent vor der Transaktion, sind sie es auch danach
- Isolation: Transaktionen beeinflussen sich nicht gegenseitig
- Dauerhaftigkeit: Wenn Transaktion erfolgreich, so ist sie in der Datenbank vorhanden

Anforderungen and Transaktionsmanagement (Folie 31)

- Concurrency Control
- !nachgucken!

Aufbau (Folie 32)

- Transaktionsmanagement sorgt für Synch der Zugriffe
- Datenbank-Cache: Lesen und Bearbeiten der Daten im DB-Cache. Schreiben geschieht später
- DB Seiten (Folie 37)

2 2.Vorlesung

Transaktion ist eine Sequenz von Operationen

Partiell geordnete Transaktionen

- Reflexiv
- Antisymmetrisch
- Transitiv

2.1 Objekt-Modell

Baumdarstellung einer Transaktion (welche Methode ruft welche Methode auf)

- Baum mit endlicher Höhe
- Innere Knoten sind Namen und Parameter von Operationen
- Blätter sind Seitenoperationen
- Besteht eine Ordung auf einer Ebene, so ist die höhere Ebene auch geordnet

2.2 Concurrency Control

Klassische Synchprobleme

Verlust von ACID Eigenschaften, wenn Transaktionen unkontrolliert ausgeführt werden.

- Lost Update Problem: Keine Kommunikation zwischen Prozessen während eines Updates
- Inconsistent Read Problem: Transaktion_1 liest während Transaktion_2 läuft $\Rightarrow$ Änderungen nicht abgeschlossen
- Dirty Read Problem: Ein Zwischenwert einer Transaktion wird für andere Transaktionen lesbar $\Rightarrow$ Abbruch der Transaktion sorgt für Fehler durch gelesenen Wert

Schedules

Modell für verschränke/parallele Ausführung von Transaktionen

- Transaction Manager entscheidet endgültig über die Reihenfolge in der Ausführung der Transaktionen verschiedener Prozesse
- Abort einer Transaktion: Rückführung aller Effekte
- Commit einer Transaktion
  - Abgeschlossen und Effekte für andere Transaktionen sichtbar
  - Muss atomar durchlaufen (ganz oder gar nicht)

Historie

- Vollständige Darstellung aller Transaktionen (mit Information über Commit und Abort)
- In der Praxis ist nur der erste Ausschnitt (Präfix) sichtbar ⇒ Schedule
- Serielle Historie: Transaktionen werden hintereinander ausgeführt (nicht parallel)
- Nicht-serielle Historie: Verschränkte Ausführung von Transaktionen

Shuffle Produkt Mischen von Transaktionen, die verschiedene Ausführungsreihenfolgen ergeben. Shuffle Produkt ist gültig, wenn

- Reihenfolge der Operationen beibehalten
- Keine Operationen hinzufügen
- Keine Operation entfernen

Aus der Menge der gültigen Produkte muss der Scheduler ein Element (eine Ausführungsmöglichkeit) wählen.

Total geordneter Schedule Entscheidungen werden auf Grund des Wissens aus den Präfixen einer Historie getroffen.

Partiell geordnete Historien Für Histoire S gilt:

- S ist vollständig: Transaktionen und Ergebnis (commit/abort)
- Eine Transaktion in S ist entweder commited oder aborted
- Ist eine Transaktion geordnet, so ist sie es auch in S
- Ordnung verschiedener Transaktionen auf dem gleichen Objekt

2.3 Reads-from Relation

2.4 FSR

2.5 View-Serialisierbarkeit VSR

3 Vorlesung

3.1 Monotonie von Historien

Eine Klasse von Historien ist monoton, wenn Schedules, die nicht in der Klasse E sind, nicht so verändert werden können, sodass sie zu E gehören. Projektionen von Historien aus E sind auch wieder in E.

3.2 Konfliktserilisierbarkeit CSR

Polynomiell entscheidbar

Konflikt Verschiedene Transaktionen greifen auf dasselbe Datenobjekt zugreifen und mindestens eine ist eine Schreiboperation.

Konfliktäquivalent Operationsanzahl und Konfliktmengen sind gleich.

Konfliktserialisierbar Ist die Reihenfolge, die sich aus den Konflikten ergibt nicht zyklisch, ist der Schedule Konfliktserialisierbar.

Beobachtung $CSR \subset VSR \subset FSR$

CSR ist monoton Wenn eine Transaktion einen Schedule nicht mehr in CSR

3.2.1 Serialisierbarkeitstheorem

- bei einer Zyklische Konfliktrelationen kann es keine konflikt-äquivalente serielle Historie geben.
- Sind die Konfliktrelationen azyklisch, ist die Historie in CSR

3.3 Kommutativität

Vertauschen von Operationen ohne die Semantik zu verändern

- Alle Operationen ohne Konflikt können bearbeitet/vertauscht werden
- Benachbarte Leser verschiedener Transaktionen können vertauscht werden
- Leser und Schreiber auf verschiedenen Objekten und von verschiedenen Transaktionen können vertauscht werden

- Schreiber auf verschiedenen Objekten und von verschiedenen Transaktionen können vertauscht werden

⇒ Eine Histore ist kommutativitäts-basiert reduzibel, wenn sie in CSR ist.

3.4 Alogrithmen für Scheduler

Aktionen eines Schedulers Für jede Transaktion muss entschieden werden:

- Output, Anfügen ans Ende des Schedules
- Reject, führt zum Fehler, also abort
- Block, funktioniert ggf, aber erst später

Arten von Schedulern

- Optimistische Scheduler
  - Vermeiden Block aktionen
  - Erst kurz vor Commit wird auf Korrektheit überprüft
- Pessimistische Scheduler
  - in der Praxis häufig verwendet, da Abort immer teuer
  - Bei Konflikt wird blockiert
  - Weniger Aborts, aber auch weniger Parallelität bis hin zum seriellen Schedule

3.4.1 Sperrende Scheduler

Lesesperre $rl_T(x)$(keine andere Transaktion kann schreiben), Schreibsperre $wl_T(x)$ (keine andere Transaktion kann lesen)

Wohlgeformtheitsregeln für Sperranfragen

1. Wenn gelesen wird muss vorher ein read lock ausgeführt worden sein (Schreiben analog)
2. Folie 190
3. S enthält keine unnötigen Sperranfragen
4. Sperren auf einem Obejkt darf nicht auf ein Objekt angewendet werden, wenn die Transaktionen in Konflikt stehen.

2-Phasen Sperrprotokoll 2PL-Scheduler Erst alle Sperren anfordern, bevor sie geschlossen werden. Nach einem Unlock, kann nicht wieder gelockt werden

Korrektheit bei Konflikt

- Bei Konflikt muss die Sperrfreigabe der ersten Transaktion vor der Anforderung der zweiten erfolgen
- Folie 200
- Konfliktgraph muss zyklenfrei sein
- Deadlock möglich, da Transaktionen endlos auf eine Sperre warten können
- Unerwartetes Einfügen von Tupeln (nur Sperren von Tupeln). Phantom-Problem. Lösung:
  - Index Locking: Zugriff auf die Tabelle wird gesperrt
  - Predicate Locking: Sperren eines logischen Ausdrucks (Ergebnis einer Anfrage)

4 Vorlesung

4.1 2Pl Protokollvarianten

Konservative C2PL

- Alle Sperren einer Transaktion wird anfangs gesetzt
- Wenn nicht alle Sperren erfolgreich gesetzt sind, werden die Konflikttransaktionen abgelehnt (nicht geblockt) $\Rightarrow$ keine Deadlocks
- Nachteil: Am Anfang müssen alle Zugriffe bekannt sein

Strikt S2PL

- Alle Schreibsperren werden erst beim Commit freigegeben (nicht Lesesperren)
- Starke SS2PL: Alle Sperren werden erst beim Transaktionsende freigegeben

4.2 Deadlocks in 2PL

- Verkantung
- Sperrkonvertierung

Lösung

1. Erkennen eines Deadlocks
   - Zyklus im Graph für Wartebeziehungen
   - Warten auf Timeout (Rückschluss auf Deadlock)
2. Eine der Zyklentransaktionen wird aborted (zB die mit am wenigsten bisher geleisteten Arbeit)

4.3 Sperrgranularität

Tradeoff mehr Sperren, weniger Synchronisationsaufwand. Weniger Sperren. mehr Parallelität.

4.3.1 Intention Reading

MGL-Protolkoll = Wie werden Anforderungen bearbeitet.

- IR
- IX
- RIX
- Miteinander im Sperrbaum kombinierbar
- Verschiedene Sperranfragen, je nach Tiefe im Baum (I-Sperren = Warnsperre)

4.3.2 Tree Locking (TL) Protokoll

Zugriff auf Tupel nur über Indexbaum (geeignet für Hierarchische Datenzugriffe).

Write Only TL WTL

1. Wohlgeformheitsregeln werden eingehalten
2. Folie 230
3.

Lock Coupling ist nicht zweiphasig. Elternknoten wird solange gesperrt, bis alle benötigten Kindknoten gesperrt sind. WTL ist Deadlockfrei

4.4 Nicht-sperrende Scheduler

Timestamp Ordering

- Keine Wartebeziehung, keine Deadlock
- Ordnung nach Zähler oder Timestamp TS(T) (Zeitstempel ist einzigartig)
- Zeitstempel der Transaktionen geben Serialisierungsreihenfolge vor
- Konflikttransationen werden nach Zeitstempel gepordnet

TO-Regel: Konflikttransaktionen sind nach ihren Timestamps (TS) geordnet.

5 Vorlesung

5.1 Mulitversionsserialisierbar: MVSR

- Wenn die Graphen äquivalent sind, haben sie auch den gleichen Konfliktgraphen

5.1.1 Versionsordnung

In einem seriellen Monoversionsschedule liegen die Zugriffe auf die Datenobjektje, wie in der Ordnung angegeben. Liegt ein MVS vor, lege eine Ordnung fest

5.1.2 Multiversions-Serialisierungsgraph MVSG

Um den Graphen aufzulösen und einen seriellen Monoversionsschedule aus einem MVS zu machen, nimm einen Knoten ohne eingehende Kanten und lösche alle ausgehenden Kanten usw. Wenn eine passende Versionsordnung gefunden ist, kann der Graph leicht gefunden werden

6 Vorlesung

6.1 Multiversion Conflict Serialisierbarkeit: MCSR

Jeder Schreiber generiert eine neue Version

6.1.1 Multiversion Conflict

- nur Read-Write Konflikte
- Umsortierung ohne Read-Write Konflikte führt zu Fehlern

6.1.2 Multiversion Conflict Graph

6.2 Mulitversionprotokolle

Protokolle, die mehr, als eine Versionen verwenden

6.2.1 Multiversions-Timestamp-Ordering MVTO

- First come, first serve.
- Leser: Letze Version vorm Zeitstempel des Lesers wird gelesen
- Schreibschritt generiert neue Version: Schreiben wir verhindert, wenns zu spät kommt (bereits eine vorherige Version gelesen)
- Versionsordnung = Zeitstempelordnung
- Commit wird erst ausgeführt, wenn alle Schreiber für die aktuelle Version Committed haben.

- Transaktion kann sich nicht selbt lesen.

6.2.2 MV2PL

Unterscheindung von Datenpbjekten

- Freigegebene Version (committed)
- aktuelle Verison (current)
- nicht freigegebene Version (uncommitted)

Nicht Commit-Operation lesen immer die vorherige Version. Immer nur einer darf ein Objekt schreiben (nur eine nicht freigegebene Version). Verzögerung des Commits, bis alle Transaktionen, die die aktuelle Version gelesen haben fertig sind. Bei Commit wird die aktuelle Version zu neuen Version für zukünfitge Operationen.

6.2.3 2V2PL

Für Recovery sinnvoll beim Abort: Before Image (BFIM), After Image (AFIM). BFIM ist aktuelle Version, AFIM die Version nach der nächsten Transaktion. Commit darf nur ausgeführt werden, wenn Andere Schreiber fertig sind. Leser können aktuelle Version lesen während, Schreiber eine neue Version schreiben

6.2.4 Read-Only Multiversions-Protokoll ROMV

Beim Beginn muss gesagt werden, ob die Transaktion schreiben wird.

- Änderungstransaktionen: S2PL oder SSPL
- Lesetransaktion: Werden nie blockiert und nicht gesperrt. Alle Änderungen, die zum **Transkationsbeginn** committed sind, werden gelesen.

Garbage collection durch Löschen nicht zuletzt freigegebenen Versionen. Lesezugriffe auf zu alte Versionen werfen dann einen Fehler.

7 Vorlesung

7.1 Korrektheit von Schedules mit Abbrüchen

Um Dirty Read Fehler zu verhindern, Verzögerung des Commits, bis alle anderen committed sind. Ein Schedule ist **recoveryfähig/rücksetzbar**, wenn voneinander lesende Transaktionen nacheinander committen. CSR impliziert nicht Recoveryfähigkeit

Vermeiden von kaskadenartigem Rücksetzen ACA WR-Konflikt. Zum Beispiel durch Halten von Schreibsperren bis zum Commit. Recoveryfähigkeit impliziert nicht das Vermeiden von kaskadenartigen Aborts

Striktheit ST WR und WW Konflikte. Lesen oder schreiben nur von NACHLESEN . ACA impliziert keine Striktheit.

Rigorose Schedules RG Alle Sperren werden bis zum Transaktionsende gehalten. ST impliziert nicht RG. Starker 2PL
$RG \subset ST \subset ACA \subset RC$
$RG \subset CSR$

7.1.1 Schedules mit Undo-Aktionen

Undo von schreibenden Transaktionen, die mit Abort enden: $w_T^{-1}(x)$ invertiert den vorherigen Schreibschritt $w_T(x)$. Beim Abort einer Transaktion, werden alle aktiven, im Schedule bereits gemachten Operationen invertiert. Die Reihenfolge der Inversen ist umgekert zur Reihenfolge der zu invertierenden Transaktionen.

7.1.2 Reduzierbarkeit RED

1. Kommutativität **CR**:
2. Undo-Regel **UR**: Benachbarte Inverse können entfernt werden.
3. Null-Aktions-Regel **NR**: Kein Effekt auf die Datenbank (Leseaktionen)
4. Ordnungs-Regel **OR**: Ungeordnete Operationen können bel geordnet werden.

Kann man mit den Regeln einen Schedule in einen seriellen Schedule umwandeln, ist er serialisierbar

Präfix-Reduzierbarkeit PRED Jedes Präfix S' von S, muss reduzierbar sein.

8 Vorlesung

8.1 Recoveryalgorithmen für das Seitenmodell

8.1.1 Fehlerklassen/Toleranz

Handling von erwartbaren Fehlern. Fehlerklassifikationen:

- Transaktionsfehler: Abort wegen fehlerhaften Aktionen, Deadlocks oder Nicht-Serialisierbarkeit. Entscheidung des Schedulers
- Systemfehler (Soft Crash): Neustart erforderlich (zB durch Stromausfall, Bedienungsfehler, CPU-Fehler)
- Gerätefehler (Hard Crash): Festplattenfehler, schwerwiegender Fehler → Verlust von Daten. Recovery durch Einspielen von Backups.

Fehlererkennung Annahme, dass das System einen Fehler erkennt.

Verfügbarkeit Wieviel Prozent der Zeit ist das System für Nutzer verfügbar: $\frac{MTTF}{MTTF+MFFT} \cdot 100$. Kürzere Reparaturzeit erhöht die Verfügbarkeit.

Redo-Recovery Arbeiten im Cache

Undo-Recovery

Restart Bist zum Punkt des Fehlers sollen alle committeten Operationen wiederholt werden. Ausführungszeit Sekunden bis Minuten

Archive-Restart, Kaltstart Wiederherstellung des committed DB state. Ausführungszeit Minuten bis Stunden

Rücksetzen Es wird angenommen. dass der Scheduler in serialisierbarer und strikter Reihenfolge weitergibt.

8.1.2 Datenbanken

stabile Datenbank nicht-flüchtiger Speicher mit Versionen der Datenbank. Zwei Arten von Änderungen:

- In-place
- Shadowing: Änderungen in einem Puffer, der mehrere Versionen eines DBobjektes enthält. Erleichtert Abort-Operationen

Datenbank-Cache Vorstufe zur stabilen Datenbank

Logging

- Physiches Log: die kompletten geänderten Seiten werden gespeichert.
- Logisches Log: nur Änderungen werden gespeichert

Seiten und Log-Einträge haben Sequenznummern. Ist die Seitensequenznummer im Cache ungleich der in der stabilen DB, gibt es Änderungen. Log-Einträge werden durch die Nummern verkettet und ergeben die Historie. Last committed value muss für jedes Objekt verfügbar sein. Konfliktreihenfolge im Schedule muss der Log-Reihenfolge entsprechen. Log muss fehlerfrei sein.

stabiles Log Log aus Änderungen an der Datenbank auf einer anderen Datenbank, damit die Daten nicht verloren gehen.

Log-Buffer Zwischenspeicher für Log-Einträge, um Einträge effizienter zu machen.

Write-Ahead-Log WAL

- **Redo/Commit-Regel:** Fehlende Änderungen von T die committed wurden müssen wiederholt werden können.
- **Log-ahead-Regel Undo-Regel:** Bevor ein Objekt in die stabile Datenbank geschrieben wird, muss zuerst ins Log geschrieben werden, damit sie Rückgängig gemacht werden kann.
- **Garbage-Collection-Regel:** Sobald ein Eintrag nicht mehr für die Recovery benötigt wird, kann sie entfernt werden.

8.1.3 Algortihmen

Verschiedene Klassifizierungen

- Undo: geschriebene Datenobjekte müssen verändert werden, weil Abort vor Commit.
- Redo: veränderte Datenobjekte werden committet, ohne dass sie in der stabilen Datenbank stehen.

- Undo/Redo-Algorithmen: System ist flexibel und Änderungen können nach Bedarf geschrieben werden.
- Undo/No-Redo-Algorithmen
- No-Undo/Redo-Algorithmen
- Np-Undo/No-Redo-Algorithmen: ideal im Fehlerfall, aber nicht im Normalfall

- Force:
- Steal: Seiten dürfen aus dem Cache auch vor dem Commit geschrieben und aus dem Cache gelöscht werden

8.1.4 Undo/Redo

9 General Stuff

- Serialiserung
- CSR
- Beweis $\in MVSR$
- Phaselocking (xPL)
- Probleme/Fehlerfälle
  - Dirty Read