Lastkontrolle

Data Contention Thrashing

- Entsteht bei zu hoher Transaktionslast und Verwendung eines pessimistischen Synchronisationsverfahrens (Sperren)
- Viele Transaktionen sind blockiert wegen Sperrkonflikte.
- Blockierte Transaktionen halten Sperren und blockieren damit andere Transaktionen.
- **Dramatischer Einbruch der Parallelität. Evtl. viele Deadlocks.**
- (Bei optimistischen Verfahren führt zu hohe Last zu Rücksetzungen.)

Lastkontrolle

- Verhindere Start von Transaktionen bei Überlast.
- Setze Transaktionen zurück bei Überlast.

Lastkontrolle

Verfahren

- Sphinx Methode
- Sisyphus Methode
- Adaptiv

Hinweis

- Verfahren werden *zusätzlich* zu Verfahren eingesetzt, die Memory Thrashing verhindern. (Wieso?)

Literatur

- Mönkeberg, Weikum: VLDB 1992.
- Weikum et al.: Information Systems 1994.

Sphinx Methode

Ansatz: Tuning Parameter M

- Etabliere BOT-Queue. Zähle Anzahl laufender Transaktionen C.
- Neue Transaktion: falls $C < M$, lasse die neue Transaktion zu ($C++$). Ansonsten hänge die neue Transaktion an BOT-Queue an.
- Commit oder Abort einer Transaktion: falls BOT-Queue nicht leer, lasse erste Transaktion der BOT-Queue zu. (Ansonsten $C--$.)

Bewertung

- M muß vom Systemadministrator bestimmt werden.
- Optimales M hängt vom Lastprofil ab:
  - Zu hoher Wert führt zu DC Thrashing.
  - Zu niedriger Wert führt zu unnötig hohen Antwortzeiten.
  - Was passiert, wenn man nur Lesetransaktionen hat?
  - Lastprofil kann sich schnell ändern.
- **Fazit:** Administratoren sind überfordert.

Sisyphus Ansatz

Ansatz

- Klassifiziere Transaktionen (z.B. nach Länge, Anzahl Lese- und Schreiboperationen).
- Etabliere BOT-Queue für jede Klasse von Transaktionen.
- Limitiere Anzahl von laufenden Transaktionen für jede Klasse wie bei der Sphinx Methode.

Bewertung

- Potentiell gut für heterogene Lastprofile.
- Wird mit schwankenden Lastprofilen auch nicht fertig.
- **Überfordert den Administrator noch mehr.** (Noch mehr Tuningparameter und Freiheitsgrade.)

Adaptive Verfahren

Ansatz

- Protokolliere *conflict ratio (C)*:

$$C = \frac{\text{Anzahl Sperren insgesamt}}{\text{Anzahl Sperren von aktiven TAs}}$$

- Im Idealfall sind alle Transaktionen aktiv und keine blockiert: $C = 1,0$.
- Lastkontrolle sorgt dafür, daß $C \leq 1,3$.
- 1,3 ist ein empirisch ermittelter Wert.

Adaptive Verfahren

Admission Policy

- Neue Transaktion wird zugelassen, wenn $C < 1.3$. Ansonsten, wird eine neue Transaktion an die BOT-Queue gehängt.
- Beim Commit oder Abort einer Transaktion wird C neu bewertet. Wenn $C < 1.3$, dann werden alle Transaktionen in der BOT-Queue zugelassen.
- **Ausnahmen:**
  - Transaktionen, die von der Lastkontrolle gecancellt wurden, bleiben mindestens $\mathcal{D}$ Zeiteinheiten in der BOT-Queue.
  - Transaktionen, die aufgrund der Deadlockauflösung zurückgesetzt wurden, bleiben mindestens solange in der BOT-Queue, bis alle anderen Transaktionen, die am Deadlock beteiligt waren, committed oder aborted wurden.
- **Verfeinerung:** Wenn Charakteristika aller Transaktionen bekannt sind, dann werden Transaktionen aus BOT-Queue selektiv nach mathematischem Modell zugelassen.
  (Details: Mönkeberg, Weikum: VLDB 1992.)
- **Frage:** Sind Commit und Abort einer Transaktion die einzigen beiden Ereignisse, die C senken?

Adaptive Verfahren

Cancellation Policy

- Cancellationentscheidungen werden gefällt, wenn eine Transaktion blockiert.
- Falls $C > 1,3$, dann setze Transaktionen zurück, bis $C < 1,3$
  (Zurücksetzungen durch Cancellation Policy triggern natürlich nicht die Admission Policy.)
- Zurückgesetzte Transaktionen kommen in die BOT-Queue. Dort bleiben sie mindestens $\mathcal{D}$ Zeiteinheiten.
- Es werden nur Transaktionen zurückgesetzt, die blockiert sind und gleichzeitig auch andere Transaktionen blockieren. (**Wieso?**)
- Heuristik: Sortiere nach

  $$\textit{Anzahl gehaltener Sperren} * \textit{Anzahl Versuche}$$

  D.h. setze Transaktionen zurück, die wenige Sperren halten und noch nicht oft zurückgesetzt wurden.
- **Frage:** Ist das blockieren einer Transaktion das einzige Ereignis, das C erhöht?

Adaptives Conflict-Ratio Verfahren

Bewertung

- Scheint besser als alles andere Bekannte zu funktionieren.
- Erfordert kaum Aufwand vom Systemadministrator.
- (Einziger Parameter ist $\mathcal{D}$ und relativ unkritisch.)

Andere Verfahren

Half-and-half Verfahren

- Verwende andere Metrik anstatt C:
$$B = \frac{\text{Anzahl aktiver Transaktionen}}{\text{Anzahl aktiver und blockierter Transaktionen}}$$
- Kritischer Schwellwert $B < 0,5$.
- **Problem:** Das Verfahren funktioniert nur richtig, wenn man zwischen *jungen* und *reifen* Transaktionen unterscheiden kann. (*reif* = TA hält mehr als 25% ihrer benötigten Sperren.) Diese Unterscheidung ist i.a. nicht möglich.

Feedback Verfahren

- Funktioniert wie Sphinx Methode, nur daß M dynamisch angepaßt wird.
- Hierzu wird der Durchsatz des Systems in einem Zeitintervall bestimmt.
- Ist der Durchsatz im Vergleich zum vorherigen Intervall gestiegen, wird M inkrementiert.
- Ist der Durchsatz im Vergleich zum vorherigen Intervall gefallen, wird M dekrementiert.
- **Problem:** Wie groß soll man das Zeitintervall wählen?

Protokollierung und Recovery

Aufgabe

- Sichere *Atomicity* und *Durability* von Transaktionen.

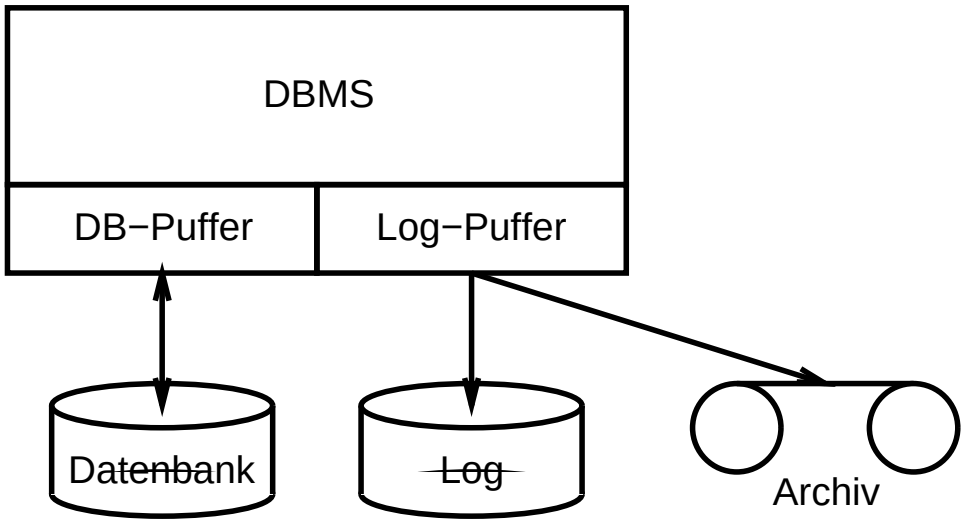
Recovery Arten

- **R1:** UNDO einer vom Benutzer (oder der Deadlockerkennung oder ...) abgebrochenen Transaktion.
- **R2:** UNDO einer Transaktion nach einem Systemabsturz (Hauptspeicherverlust).
- **R3:** REDO einer Transaktion nach einem Systemabsturz (Hauptspeicherverlust).
- **R4:** Wiederherstellung der Datenbasis nach Festplattenverlust.

Annahmen

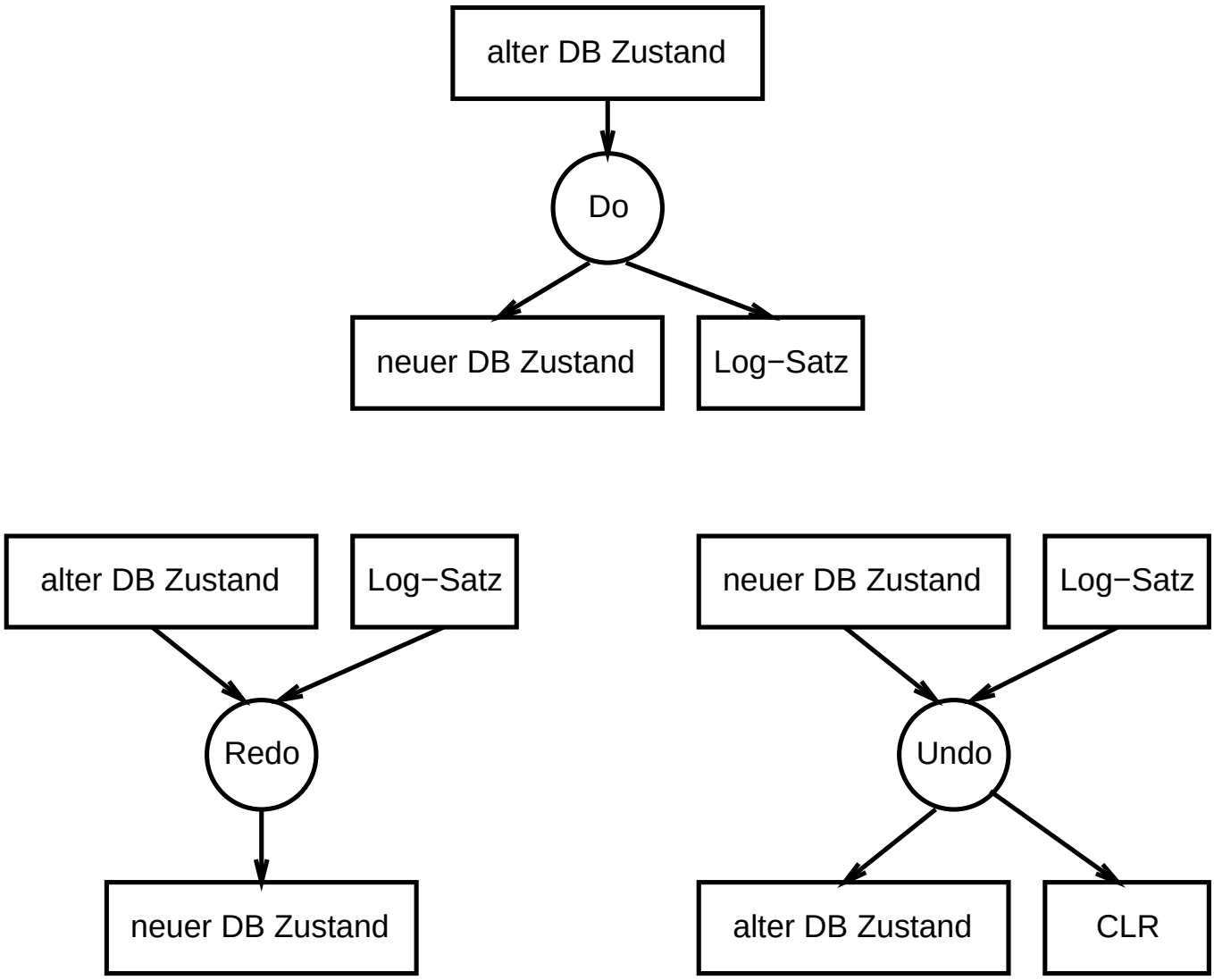
- Steal, update in place, No Force
- Fast jedes System macht es so.
- Andere Annahmen vereinfachen die Recovery.

Architektur



- Protokollsätze (Log-Daten) werden gepuffert wie andere Daten auch (allerdings mit besonderen Regeln).
- Protokolldaten werden auf separate Platte und evtl. auf Band geschrieben.
- Archivkopie für Plattenverlust wird auf Band geschrieben.

Do-Redo-Undo Prinzip



Protokollierung (Logging)

Physisch

- Zustandslogging: ⟨ Before Image, After Image ⟩
- Übergangslogging (Diffing):

  ⟨ Before Image XOR After Image ⟩

Logisch

- ⟨ Änderungsoperation ⟩
- Änderungsoperation muß für UNDO invertierbar sein.

Physiologisch

- physisch für Änderungen innerhalb eines Objektes
- logisch für Änderungen innerhalb einer Seite (z.B. Einfügungen, Verschiebungen, Löschungen)

Granularität des Loggings

Seite

- leicht zu implementieren
- große Protokollsätze
- verträgt sich nicht mit fein-granularer Sperrverwaltung
  (siehe Übung)
- wird nur in verteilten (Client/Server) Systemen eingesetzt

Objekte

- kleine Protokollsätze
- weniger IO, besseres Puffern möglich, Group-Commit möglich (siehe später)
- verträgt sich mit fein-granularer Sperrverwaltung
- macht jedes gängige (zentrale) System

Log Sequence Numbers (LSN)

Grundlagen

- jeder Protokollsatz hat eindeutige Adresse: LSN
- LSNs wachsen streng monoton
- DB-Seiten enthalten im Seitenkopf LSN der letzten Änderung

Nutzung im Fehlerfall

- **REDO** nur erforderlich, wenn

  Page-LSN $<$ Log-LSN
- **UNDO** nur erforderlich, wenn

  Page-LSN $\geq$ Log-LSN

Aufbau der Log-Datei

- Log-Sätze werden ans Ende der Log-Datei geschrieben.
- Arten von Log-Sätzen:
  - BOT, Commit, Abort
  - UNDO/REDO Sätze
  - Checkpoint Sätze
  - CLR
- Log-Sätze einer Transaktion werden rückwärts verkettet
  (einfaches UNDO)
- Log-Daten sind nur für eine begrenzte Zeit relevant:
  - Organisation der Log-Datei als Ringpuffer.

Prinzipien

WAL Prinzip (für UNDO)

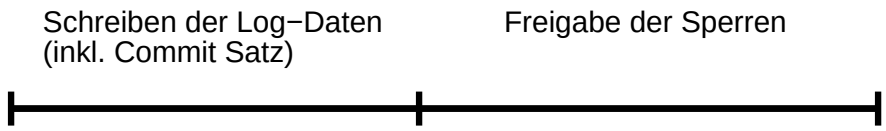
- Bevor eine DB-Seite auf Platte geschrieben wird (z.B. aufgrund von Seitenersetzung) müssen alle Log-Einträge der Seite in die Log-Datei auf Platte geschrieben werden.
- (Diese Forderung ergibt sich aus den Steal und Update-in-place Annahmen.)

Commit Regel (für REDO)

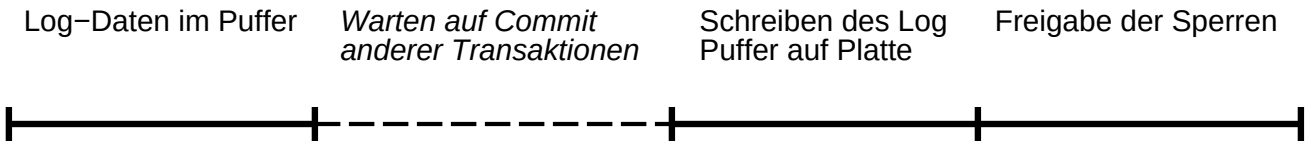
- Bei Commit einer Transaktion müssen alle Log Records der Transaktion auf in die Log-Datei auf Platte geschrieben werden.
- (Diese Forderung ergibt sich aus der No-force Annahme.)

Commit Verarbeitung

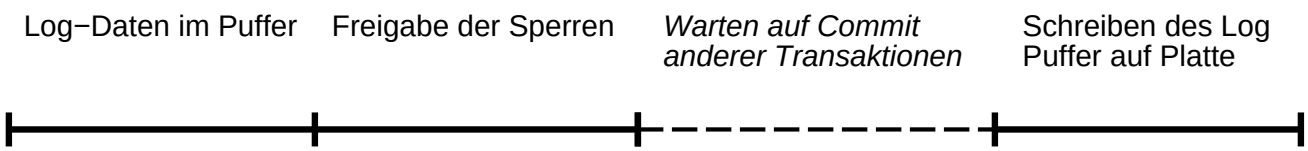
Reguläres Zweiphasen Commit (zentral):



Group Commit (zentral):



Präcommit (zentral):



Frage: Wieso funktioniert Präcommit in zentralen Systemen? Funktioniert es auch in verteilten Systemen?

Sicherungspunkte

Transaktionskonsistente Sicherungspunkte

- Ausschreiben des Sicherungspunktes verzögern bis zum Ende aller aktiven Änderungstransaktionen.
- Neue Transaktionen müssen warten bis Sicherungspunkt beendet ist.
- Crash Recovery startet bei Sicherungspunkt.

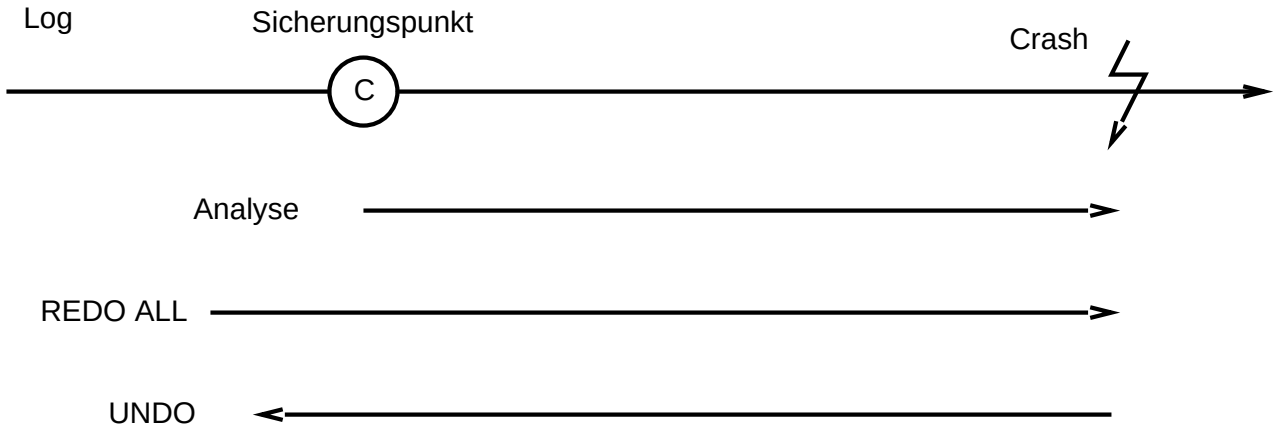
Aktionskonsistente Sicherungspunkte

- keine Änderungen während des Sicherungspunktes.
- Crash-Recovery wird nicht durch Sicherungspunkt begrenzt.

Fuzzy Sicherungspunkte

- Gar keine Totzeit während des Sicherungspunktes.
- Noch schlechtere Begrenzung der Crash-Recovery durch den Sicherungspunkt.

Crash Recovery (ARIES)



- Aufsatzpunkt des Redo-Laufs hängt von Art des Sicherungspunktes ab.
- Zielpunkt des Undo-Laufs hängt auch von Art des Sicherungspunktes ab.
- N.B.: Beim Undo-Lauf müssen CLR (Compensation-Log Records) gesetzt werden, um *Redo des Undo* im Falle eines Crashes während der Recovery zu gewährleisten.
- **Frage:** Wieso macht man bei ARIES *REDO ALL* und dann erst ein *UNDO*?

Archivierung, Geräterecovery

Motivation

- Der Hausmeister hat den Schlüssel zum Rechenzentrum verloren. Bei der Sprengung der Tür stürzt der Rechner ab und alle Platten gehen kaputt.
- Die Datenbasis muß von Band neu eingespielt werden.
- **Wichtig:** Sorgt dafür, daß Ihr ein Backupband zu Hause unter Eurem Kopfkissen liegen habt.

Menü

- Fuzzy Dumps
- Transaktionskonsistente Dumps
- Inkrementelle Dumps

Fuzzy Dumps

- Archivkopie wird im laufenden Betrieb gezogen.
- Zu Beginn wird aktuelle LSN der Logdatei auf Band notiert.
- **Variante 1:** Völlig ohne Lesesperren (nur Semaphore)
  - keine Beeinträchtigung des laufenden Betriebs; schneller Dump
  - REDO und UNDO auf Archivkopie notwendig
- **Variante 2:** Kurze Lesesperren
  - Dumplauf muß auf Ende von Schreibtransaktionen warten
  - REDO aber kein UNDO auf Archivkopie notwendig
  - (Level 1 Isolation)

Transaktionskonsistenter Dump

Ziele

- Dump sollte ohne REDO oder UNDO einspielbar sein.
- Man benötigt keine zusätzlichen Protokollinfo auf Band.
- Nur geringer zusätzlicher Aufwand und Störung des Betriebs.
- Portabler Ansatz.

Black & White Verfahren

- Paint-Bit pro Seite:
  - *white:* Seite wurde noch nicht überprüft
  - *black:* Seite wurde verarbeitet
- Schreibprozeß färbt alle weißen Seiten schwarz und schreibt geänderte Seiten in Archivkopie:
  1. lock page
  2. write page to archive copy
  3. change page color
  4. release page lock

Black & White Verfahren

VORSICHT:

- *Graue* Transaktionen, die sowohl weiße als auch schwarze Objekte geändert haben, werden zurückgesetzt.

Mögliche Abhilfen

- Während des Dump-Prozeßes werden Before-Images der weißen Seiten aufgehoben und es werden nur solche Before-Images gedumped.
- Oder: Zugriffe auf weiße Objekte von Transaktionen werden verboten.
- Oder: ...

Inkrementeller Dump

Ziele

- Schreibe nur Seiten auf Band, die sich geändert haben.
- Spare Platz.
- Reduziere "Fenster" für Backup (i.e., Laufzeit des Dump).
- (Abhängig vom Szenario kann hierdurch allerdings Laufzeit der Crashrecovery größer werden.)

Verfahren

- Archivierungsbits in den Datenbankseiten.
- Merken modifizierter Seiten in separaten Strukturen.

N.B.: Wöchentlich voller (fuzzy oder transaktionskonsistenter) Dump plus täglich inkrementeller Dump ist häufig die beste Lösung. (So wird es z.B. an unserem Lehrstuhl gemacht.)