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?