Gruppe 22 / 48

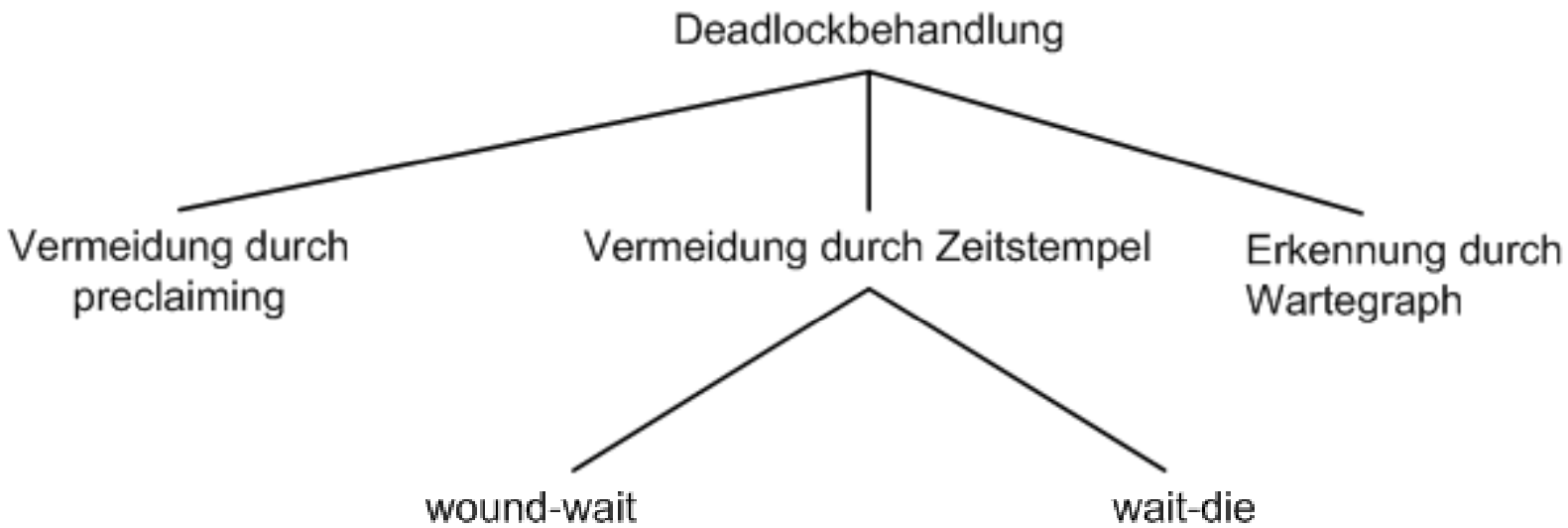
Tutorübung zu Einsatz und Realisierung von Datenbanksystemen (SS 18)

Michael Schwarz

Institut für Informatik
Technische Universität München

24.04 / 26.04.2018

Klassifikation der Deadlock-Behandlungsverfahren



Verklemmungsvermeidung durch Zeitstempel

- Jeder Transaktion wird ein eindeutiger Zeitstempel (TS) zugeordnet
- ältere TAs haben einen kleineren Zeitstempel als jüngere TAs
- TAs dürfen nicht mehr „bedingungslos“ auf eine Sperre warten

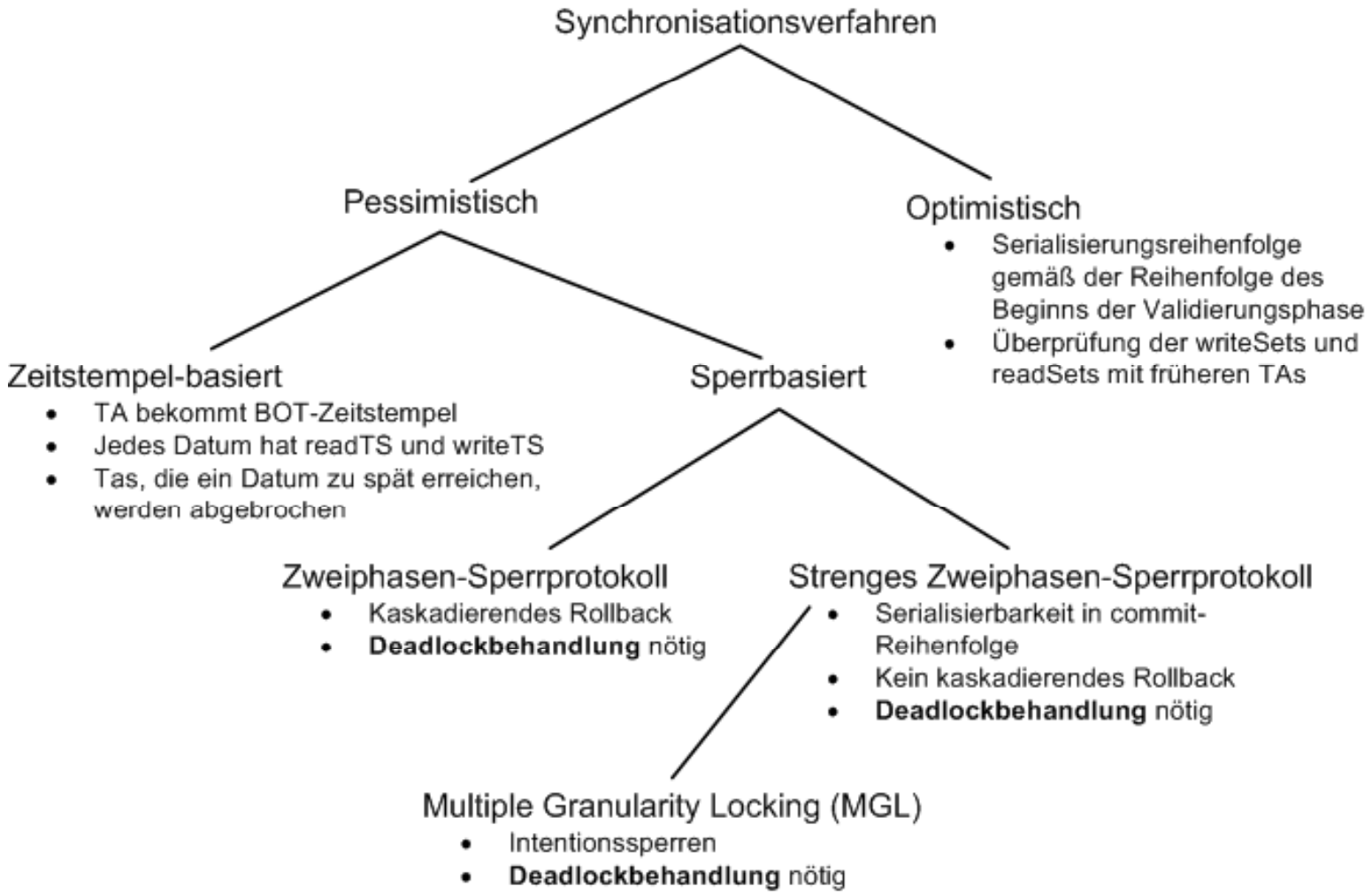
wound-wait Strategie

- T_1 will Sperre erwerben, die von T_2 gehalten wird.
- Wenn T_1 älter als T_2 ist, wird T_2 abgebrochen und zurückgesetzt, so dass T_1 weiterlaufen kann.
- Sonst wartet T_1 auf die Freigabe der Sperre durch T_2.

wait-die Strategie

- T_1 *will Sperre erwerben, die von* T_2 gehalten wird.
- Wenn T_1 älter als T_2 ist, wartet T_1 auf die Freigabe der Sperre.
- Sonst wird T_1 abgebrochen und zurückgesetzt.

Klassifikation der Synchronisationsverfahren



Optimistische Synchronisation

1. *Lesephase:*
   - In dieser Phase werden alle Operationen der Transaktion ausgeführt – also auch die Änderungsoperationen.
   - Gegenüber der Datenbasis tritt die Transaktion in dieser Phase aber nur als Leser in Erscheinung, da alle gelesenen Daten in lokalen Variablen der Transaktion gespeichert werden.
   - alle Schreiboperationen werden zunächst auf diesen lokalen Variablen aufgeführt.
2. *Validierungsphase:*
   - In dieser Phase wird entschieden, ob die Transaktion möglicherweise in Konflikt mit anderen Transaktionen geraten ist.
   - Dies wird anhand von Zeitstempeln entschieden, die den Transaktionen in der Reihenfolge zugewiesen werden, in der sie in die Validierungsphase eintreten.
3. *Schreibphase:*
   - Die Änderungen der Transaktionen, bei denen die Validierung positiv verlaufen ist, werden in dieser Phase in die Datenbank eingebracht.

Validierung bei der Snapshot Isolation (SI)

Vorsicht: SI garantiert nicht die Serialisierbarkeit – wird aber heute oft für hoch-skalierende Systeme verwendet

Wir wollen eine Transaktion T_j validieren. Die Validierung ist erfolgreich falls für **alle** älteren Transaktionen T_a – also solche die früher ihre Validierung abgeschlossen haben – eine der beiden folgenden Bedingungen gelten:

1. T_a war zum Beginn der Transaktion T_j schon abgeschlossen – einschließlich der Schreibphase.
2. Die Menge der von T_a geschriebenen Datenelemente, genannt $WriteSet(T_a)$ enthält keine Elemente der Menge der ~~gelesenen~~ geschriebenen Datenelemente von T_j, genannt $WriteSet(T_j)$. *Es muss also gelten:*

$$\sout{ReadSet}\ WriteSet(T_a) \cap WriteSet(T_j) = \varnothing$$

Validierung bei der optimistischen Synchronisation

Vereinfachende Annahme: Es ist immer nur eine TA in der Validierungsphase!

Wir wollen eine Transaktion T_j validieren. Die Validierung ist erfolgreich falls für **alle** älteren Transaktionen T_a – also solche die früher ihre Validierung abgeschlossen haben – eine der beiden folgenden Bedingungen gelten:

1. T_a war zum Beginn der Transaktion T_j schon abgeschlossen – einschließlich der Schreibphase.
2. Die Menge der von T_a geschriebenen Datenelemente, genannt $WriteSet(T_a)$ enthält keine Elemente der Menge der gelesenen Datenelemente von T_j, genannt $ReadSet(T_j)$. *Es muss also gelten:*

$$WriteSet(T_a) \cap ReadSet(T_j) = \varnothing$$

Zeitstempel-basierende Synchronisation

Jedem Datum A in der Datenbasis werden bei diesem Synchronisationsverfahren zwei Marken zugeordnet:

1. *readTS(A):*
2. *writeTS(A):*

Synchronisationsverfahren

- T_i will A lesen, also $r_i(A)$
  - Falls $TS(T_i) < writeTS(A)$ gilt, haben wir ein Problem:
    - Die Transaktion T_i ist älter als eine andere Transaktion, die A schon geschrieben hat.
    - Also muss T_i zurückgesetzt werden.
  - Anderenfalls, wenn also $TS(T_i) \geq writeTS(A)$ gilt, kann T_i ihre Leseoperation durchführen und die Marke $readTS(A)$ wird auf $max(TS(T_i), readTS(A))$ gesetzt.

Zeitstempel-basierende Synchronisation

Synchronisationsverfahren

- T_i will A schreiben, also $w_i(A)$
  - Falls $TS(T_i) < readTS(A)$ gilt, gab es eine jüngere Lesetransaktion, die den neuen Wert von A, den T_i gerade beabsichtigt zu schreiben, hätte lesen müssen. Also muss T_i zurückgesetzt werden.
  - Falls $TS(T_i) < writeTS(A)$ gilt, gab es eine jüngere Schreibtransaktion. D.h. T_i beabsichtigt einen Wert einer jüngeren Transaktion zu überschreiben. Das muss natürlich verhindert werden, so dass T_i auch in diesem Fall zurückgesetzt werden muss.
  - Anderenfalls darf T_i das Datum A schreiben und die Marke $writeTS(A)$ wird auf $TS(T_i)$ gesetzt.