Transaktionale Informationssysteme

-

4. Synchronisation - Algorithmen II

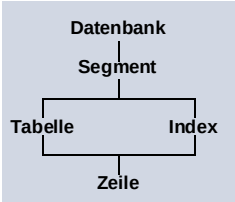
Norbert Ritter
Datenbanken und Informationssysteme
vsis-www.informatik.uni-hamburg.de

Hierarchische Sperrverfahren (1)

- Sperrgranulat
  - bestimmt Parallelität/Aufwand
  - feines Granulat reduziert Sperrkonflikte, jedoch sind viele Sperren anzufordern und zu verwalten
  - hierarchische Verfahren erlauben Flexibilität bei Wahl des Granulates ('multi-granularity locking')
  - z. B. Synchronisation langer TA auf Tabellenebene
  - oder kurzer TA auf Zeilenebene
  - kommerzielle DBS unterstützen zumeist mindestens 2-stufige Objekthierarchie, z.B.
    - Seite – Segment
    - Satztyp (Tabelle) – Satzausprägung (Tupel)

Hierarchische Sperrverfahren (2)

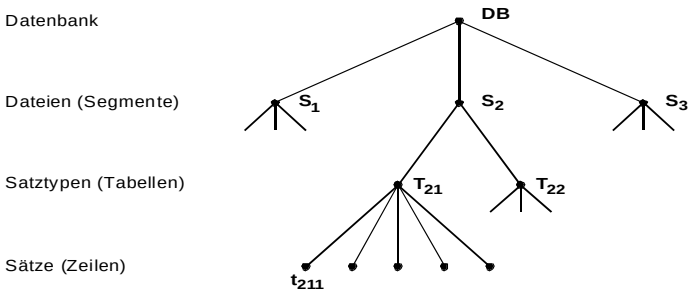
- Verfahren nicht auf reine Hierarchien beschränkt, sondern auch auf halbgeordnete Objektmengen erweiterbar (siehe auch objektorientierte DBS)



- Verfahren erheblich komplexer als einfache Sperrverfahren (mehr Sperrmodi, Konversionen, Deadlock-Behandlung, ...)

Hierarchische Sperrverfahren (3)

- Beispiel einer Sperrhierarchie



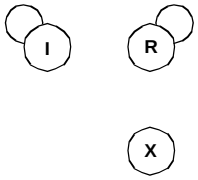
Aufwand zum Sperren von

- 1 Satz : 3 + 1
- k Sätzen : 3 + k
- 1 Satztyp : 2 + 1

Hierarchische Sperrverfahren (4)

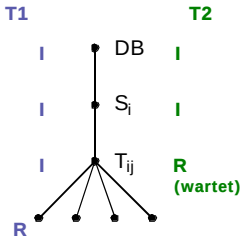
- Anwartschaftssperren
  - durch R- und X-Sperre werden alle Nachfolgerknoten implizit mitgesperrt → Einsparungen möglich
  - alle Vorgängerknoten sind ebenfalls zu sperren, um Unverträglichkeiten zu vermeiden
  - Verwendung von Anwartschaftssperren ('intention locks')
  - allgemeine Anwartschaftssperre: I-Sperre

	I	R	X
I	+	-	-
R	-	+	-
X	-	-	-

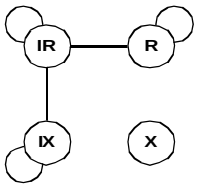


Hierarchische Sperrverfahren (5)

- Anwartschaftssperren (Forts.)
  - allgemeine Anwartschaftssperre – Beispiel
  - Unverträglichkeit von I- und R-Sperren
    - zu restriktiv!
  - Lösung (?): zwei Arten von Anwartschaftssperren: IR und IX

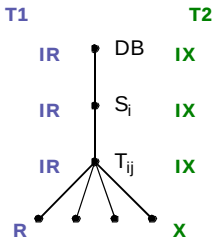


	IR	IX	R	X
IR	+	+	+	-
IX	+	+	-	-
R	+	-	+	-
X	-	-	-	-



Hierarchische Sperrverfahren (6)

- Anwartschaftssperren (Forts.)
  - IR und IX - Beispiel
  - IR-Sperre (intent read), falls auf untergeordneten Objekten nur lesend zugegriffen wird, sonst IX-Sperre

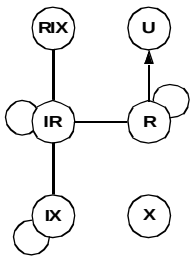


  - weitere Verfeinerung: RIX = R + IX
    - für den Fall, in dem alle Sätze eines Satztyps gelesen aber nur einige geändert werden sollen
      - X-Sperre auf Satztyp wäre zu restriktiv
      - IX-Sperre auf Satztyp würde Sperren jedes Satzes verlangen
    - sperrt das Objekt in R-Modus und verlangt
    - X-Sperren auf tieferer Hierarchieebene nur für zu ändernde Objekte

Hierarchische Sperrverfahren (7)

- Anwartschaftssperren (Forts.)
  - Vollständiges Protokoll der Anwartschaftssperren
    - RIX gibt ein Leserecht auf den Knoten und seine Nachfolger; weiterhin ist damit das Recht verbunden, auf Nachfolger-Knoten IX, U und X-Sperren anzufordern
    - U gewährt ein Leserecht auf den Knoten und seine Nachfolger; dieser Modus repräsentiert die Absicht, den Knoten in der Zukunft zu verändern; bei Änderung Konversion U → X, sonst U → R

	IR	IX	R	RIX	U	X
IR	+	+	+	+	-	-
IX	+	+	-	-	-	-
R	+	-	+	-	-	-
RIX	+	-	-	-	-	-
U	-	-	+	-	-	-
X	-	-	-	-	-	-

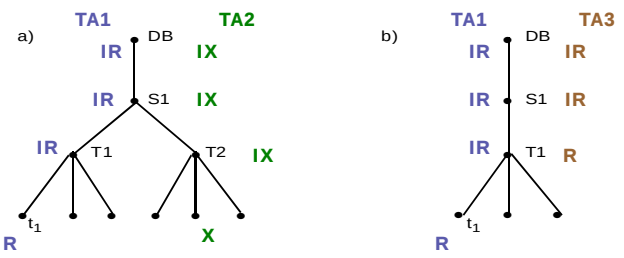


Hierarchische Sperrverfahren (8)

- Anwartschaftssperren (Forts.)
  - Vollständiges Protokoll der Anwartschaftssperren (Forts.)
    - Sperrdisziplin erforderlich
      - Sperranforderungen von der Wurzel zu den Blättern
      - bevor T eine R- oder IR-Sperre für einen Knoten anfordert, muss sie für alle Vorgängerknoten IX- oder IR-Sperren besitzen
      - bei einer X-, U-, RIX- oder IX-Anforderung müssen alle Vorgängerknoten in RIX oder IX gehalten werden
      - Sperrfreigaben von den Blättern zu der Wurzel
      - bei EOT sind alle Sperren freizugeben

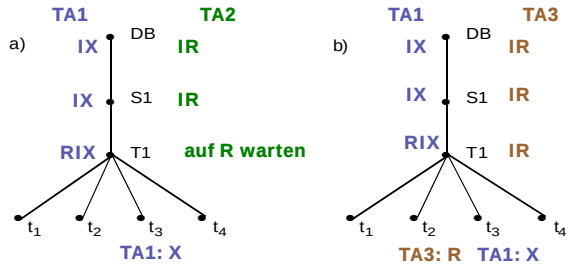
Hierarchische Sperrverfahren (9)

- Anwartschaftssperren (Forts.)
  - Vollständiges Protokoll der Anwartschaftssperren (Forts.)
    - Beispiel
      - IR- und IX-Modus
        - TA1 liest t_1 in T1
        - TA2 ändert Zeile in T2 (a)
        - TA3 liest T1 (b)



Hierarchische Sperrverfahren (10)

- Anwartschaftssperren (Forts.)
  - Vollständiges Protokoll der Anwartschaftssperren (Forts.)
    - Beispiel
      - RIX-Modus
        - TA1 liest alle Zeilen von T1 und ändert t_3
        - TA2 liest T1 (a)
        - TA3 liest t_2 in T1 (b)

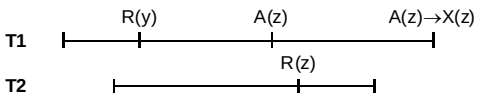


Sperrverfahren mit Versionen (1)

- RAX
  - Kompatibilitätsmatrix

	R	A	X
R	+	⊕	-
A	⊕	-	-
X	-	-	-

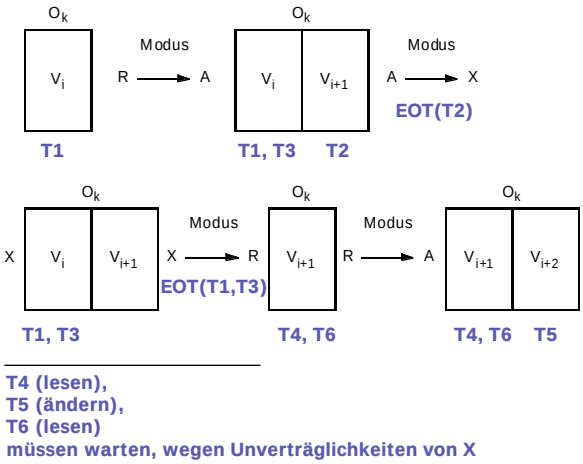
  - Ablaufbeispiel



RAX: T2 → T1

Sperrverfahren mit Versionen (2)

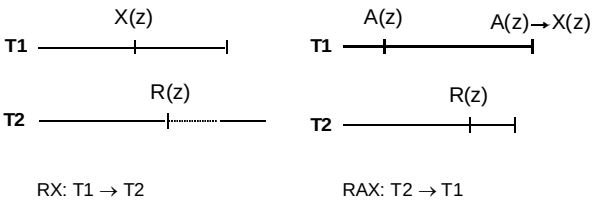
- RAX (Forts.)
  - Änderungen erfolgen in temporärer Objektkopie



T4 (lesen),
T5 (ändern),
T6 (lesen)
müssen warten, wegen Unverträglichkeiten von X

Sperrverfahren mit Versionen (3)

- RAX (Forts.)
  - Eigenschaften von RAX
    - paralleles Lesen der gültigen Version wird zugelassen
    - Schreiben wird nach wie vor serialisiert (A-Sperre)
    - bei EOT Konversion der A- nach X-Sperren, ggf. auf Freigabe von Lesesperren warten (Deadlock-Gefahr)
    - höhere Parallelität als beim RX-Verfahren, jedoch i. allg. andere Serialisierungsreihenfolge:



RX: T1 → T2 RAX: T2 → T1

Sperrverfahren mit Versionen (4)

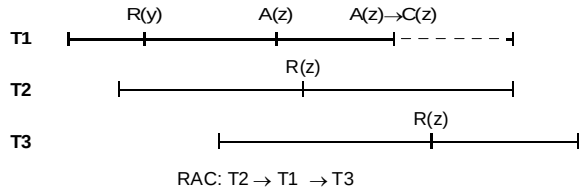
- RAX (Forts.)
  - Nachteile
    - bei TA-Mix von langen Lesern und kurzen Schreibern auf gemeinsamen Objekten bringt RAX keinen großen Vorteil
      - neue Version wird für neu ankommende Leser erst verfügbar, wenn alte Version aufgegeben werden kann
      - starke Behinderungen von Update-TA durch (lange) Leser möglich

Sperrverfahren mit Versionen (5)

- RAC
  - Kompatibilitätsmatrix

	R	A	C
R	+	+	⊕
A	+	-	-
C	⊕	-	-

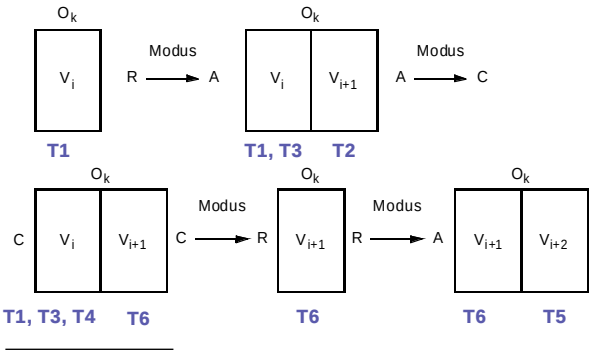
  - Ablaufbeispiel



RAC: T2 → T1 → T3

Sperrverfahren mit Versionen (6)

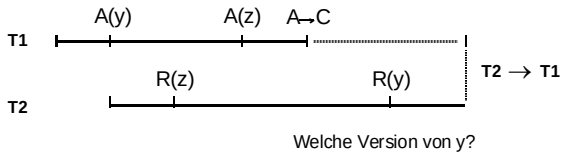
- RAC (Forts.)
  - Änderungen erfolgen ebenfalls in temporärer Objektkopie



für T4 und T6 (lesen) kann sofort geeignete Version bestimmt werden, T5 (ändern) muss warten, da nur 2 Versionen

Sperrverfahren mit Versionen (7)

- RAC (Forts.)
  - Eigenschaften von RAC
    - Änderungen werden nach wie vor serialisiert (A-Sperre erforderlich)
    - bei EOT Konversion von A- nach C-Sperre
    - maximal 2 Versionen, da C-Sperren mit sich selbst und mit A-Sperren unverträglich sind
    - C-Sperre zeigt Existenz zweier gültiger Objektversionen an



    - kein Warten auf Freigabe von Lesesperren auf alter Version (R- und C-Modus sind verträglich)

Sperrverfahren mit Versionen (8)

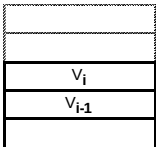
- RAC (Forts.)
  - Nachteile
    - RAC ist nicht chronologieerhaltend
    - Verwaltung komplexer Abhängigkeiten (z. B. über Abhängigkeitsgraphen) komplexere Sperrverwaltung
    - Leseanforderungen bewirken nie Blockierung/Rücksetzung, jedoch Auswahl der „richtigen“ Version erforderlich
    - Änderungs-TA, die auf C-Sperre laufen, müssen warten, bis alle Leser der alten Version beendet, weil nur 2 Versionen
  - Abhilfe: allgemeines Mehrversionen-Verfahren

Sperrverfahren mit Versionen (9)

- Mehrversionenverfahren
  - Prinzip
    - Änderungs-TA erzeugen neue Objektversionen
      - es kann immer nur eine neue Version pro Objekt erzeugt werden
      - sie wird bei EOT der TA freigegeben
    - Lese-TA sehen den bei ihrem BOT gültigen DB-Zustand
      - sie greifen immer auf die jüngsten Objektversionen zu, die bei ihrem BOT freigegeben waren
      - sie setzen und beachten keine Sperren
      - es gibt keine Blockierungen und Rücksetzungen für Lese-TA, dafür ggf. Zugriff auf veraltete Objektversionen

Sperrverfahren mit Versionen (10)

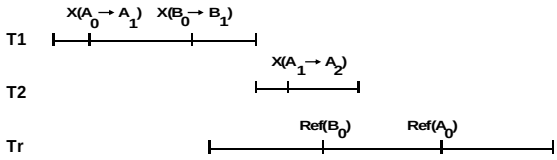
- Mehrversionenverfahren (Forts.)
  - Beispiel für Objekt O_k
    - Zeitliche Reihenfolge der Zugriffe auf O_k



T_j (BOT)	→	V_i (aktuelle Version)
T_m(X)	→	Erzeugen V_{i+1}
T_n(X)	→	Verzögern bis T_m(EOT)
T_m(EOT)	→	Freigeben V_{i+1}
T_n(X)	→	Erzeugen V_{i+2}
T_j (Ref)	→	V_i
T_n(EOT)	→	Freigeben V_{i+2}

Sperrverfahren mit Versionen (11)

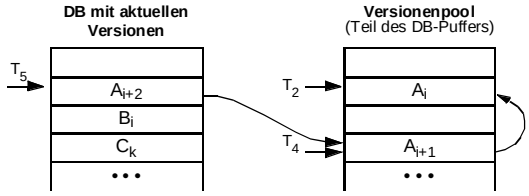
- Mehrversionenverfahren (Forts.)
  - Beispiel



  - Konsequenz
    - deutlich weniger Synchronisationskonflikte
      - Lese-TA werden bei der Synchronisation nicht mehr berücksichtigt
      - Änderungs-TA werden untereinander über ein allgemeines Verfahren (Sperren, OCC, . . .) synchronisiert

Sperrverfahren mit Versionen (12)

- Mehrversionenverfahren (Forts.)
  - zusätzlicher Speicher- und Wartungsaufwand
    - Versionenpoolverwaltung, Garbage Collection
    - Auffinden von Versionen



    - Speicherplatzoptimierung: Versionen auf Satzebene, Einsatz von Komprimierungstechniken
  - Einsatz in einigen kommerziellen DBVS (z.B. Oracle)

Prädikatsperren (1)

- Logische Sperren
  - Minimaler Sperrbereich durch geeignete Wahl des Prädikats
  - Verhütung des Phantomproblems
  - Eleganz
- Form
  - LOCK (R, P, a), UNLOCK (R, P)
    - R: Relationenname
    - P: Prädikat
    - a ∈ {read, write}
  - *Lock (R, P, write)* sperrt alle möglichen Tupeln von R exklusiv, die Prädikat P erfüllen

Eswaran, K.P. et al.: The notions of consistency and predicate locks in a data base system. in: Commm. ACM 19:11, 1976, 624-633

Prädikatsperren (2)

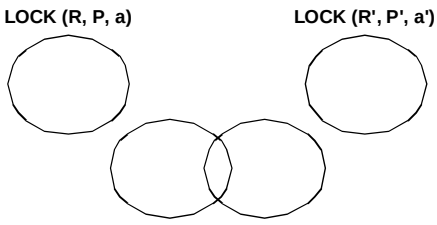
- Beispiel:

T1:		T2:	
	LOCK(R1, P1, read)		. . .
	LOCK(R2, P2, write)		LOCK(R2, P3, write)
	LOCK(R1, P5, write)		LOCK(R1, P4, read)
	. . .		. . .

- Problem: Konfliktfeststellung
  - im allgemeinen Fall rekursiv unentscheidbar, selbst mit eingeschränkten arithmetischen Operatoren
  - entscheidbare Klasse: einfache Prädikate der Form (A Θ Wert) {$\wedge$, $\vee$} (. . .

Prädikatsperren (3)

- Entscheidungsprozedur



1. Wenn R $\neq$ R', kein Konflikt
2. Wenn a = read und a' = read, kein Konflikt
3. Wenn P(t) $\wedge$ P'(t) = TRUE für irgendein t, dann besteht ein Konflikt

- Beispiel

T1:	T2:
LOCK (Pers, Alter > 50, read)	LOCK (Pers, Pnr = 4711, write)

Prädikatsperren (4)

- Nachteile
  - Erfüllbarkeitstest: aufwendige Entscheidungsprozedur; i.d.R. mit vielen Prädikaten (N > 100); wird in innerer Schleife des Lock-Mgr. häufig aufgerufen)
  - pessimistische Entscheidungen → Einschränkung der Parallelität (es wird auf Erfüllbarkeit getestet !)
  - Einsatz nur bei deskriptiven Sprachen!
  - Sonderfall: P=TRUE entspricht einer Relationensperre → große Sperrgranulate, geringe Parallelität

Prädikatsperren (5)

- Effizientere Implementierung: Präzisionssperren
  - nur gelesene Daten werden durch Prädikate gesperrt
  - für aktualisierte Tupel werden Schreibsperren gesetzt
  - kein Disjunktheitstest für Prädikate mehr erforderlich, sondern lediglich zu überprüfen, ob Tupel ein Prädikat erfüllt
  - Datenstrukturen
    - Prädikatliste: Lesesperren laufender TA werden durch Prädikate beschrieben
      (Pers: Alter > 50 and Beruf = 'Prog.')
      (Pers: Pname = 'Meier' and Gehalt > 50000)
      (Abt: Anr = K55)
      . . .
    - Update-Liste: enthält geänderte Tupel laufender TA
      (Pers: 4711, 'Müller', 30, 'Prog.', 70000)
      (Abt: K51, 'DBS', . . .)
      . . .

J.R. Jordan, J. Banerjee, R.B. Batman: Precision Locks, in: Proc. ACM SIGMOD, 1981, 143-147

Prädikatsperren (6)

- Präzisionssperren (Forts.)
  - Leseanforderung (Prädikat P):
    - für jedes Tupel der Update-Liste ist zu prüfen, ob es P erfüllt
    - wenn ja → Sperrkonflikt
  - Schreibanforderung (Tupel T):
    - für jedes Prädikat P der Prädikatliste ist P(T) zu prüfen
    - wenn T keines erfüllt → Schreibsperre wird gewährt

Semantische Synchronisation (1)

- Idee
  - Erhöhung der Parallelität durch Einführung kommutativer („semantischer“) DB-Operationen als Einheit für die Synchronisation
  - Synchronisation erfolgt auf abstrakterer Ebene (Objektebene)
  - Realisierung muss auf niedrigerer Ebene Korrektheit gewährleisten (z. B. durch Escrow-Verfahren)

Semantische Synchronisation (2)

- Beispiel
  - Zwei Kontobuchungen auf Konten K1 und K2 durch die TA T1 und T2, die kommutative Operationen „erhöhe um x“ und „vermindere um x“ verwenden, könnten z. B. in folgenden Reihenfolgen ausgeführt werden:

Schedule S1

1. erhöhe (T1,K1,x1)
2. vermindere (T1,K2,x1)
3. erhöhe (T2,K1,x2)
4. vermindere (T2,K2,x2)

Schedule S2

1. erhöhe (T1,K1,x1)
2. erhöhe (T2,K1,x2)
3. vermindere (T2,K2,x2)
4. vermindere (T1,K2,x1)

Schedule S1 wäre auch mit rein syntaktischen Verfahren (r/w) erzeugbar:

$<\underbrace{r_1(K_1)\ w_1(K_1)}_{\text{erhöhe}(T_1,K_1,x_1)}\ \underbrace{r_1(K_2)\ w_1(K_2)}_{\text{vermindere}(T_1,K_2,x_1)}\ \underbrace{r_2(K_1)\ w_2(K_1)}_{\text{erhöhe}(T_2,K_1,x_2)}\ \underbrace{r_2(K_2)\ w_2(K_2)}_{\text{vermindere}(T_2,K_2,x_2)}>$

Schedule S2 hingegen wäre mit rein syntaktisch arbeitenden Verfahren nicht erzeugbar

‚High-Traffic‘-Synchronisation (1)

- Synchronisation von High-Traffic-Objekten
  - High-Traffic-Objekte
    - meist numerische Felder mit aggregierten Informationen, z. B.
      - Anzahl freier Plätze
      - Summe aller Kontostände
  - Behandlung
    - Einfachste Lösung der Sperrprobleme
      - Vermeidung solcher Attribute beim DB-Entwurf
    - Alternative
      - Nutzung von semantischem Wissen zur Synchronisation wie Kommutativität von Änderungsoperationen auf solchen Feldern
      - Beispiel: Inkrement/Dekrement

	R	X	Inc/Dec
R	+	-	-
X	-	-	-
Inc/Dec	-	-	+

‚High-Traffic'-Synchronisation (2)

- Escrow-Ansatz
  - Deklaration von High-Traffic-Objekten als Escrow-Felder
  - Benutzung spezieller Operationen
  - Anforderung einer bestimmten Wertemenge
    - IF **ESCROW (field=F1, quantity=C1, test=(condition))**
      THEN 'continue with normal processing'
      ELSE 'perform exception handling'
  - Benutzung der reservierten Wertmengen:
    - **USE (field=F1, quantity=C2)**
  - optionale Spezifikation eines Bereichstest bei Escrow-Anforderung
  - wenn Anforderung erfolgreich ist, kann Prädikat nicht mehr nachträglich invalidiert werden

P. O'Neil: The Escrow Transactional Method,
in: ACM Trans. on Database Systems 11: 4, 1986, 405-430

‚High-Traffic'-Synchronisation (3)

- Escrow-Ansatz (Forts.)
  - aktueller Wert eines Escrow-Feldes
    - ist unbekannt, wenn laufende TA Reservierungen angemeldet haben
    - Führen eines Werteintervalls, das alle möglichen Werte nach Abschluss der laufenden TA umfasst
    - für Wert Q_k des Escrow-Feldes k gilt
      - $LO_k \leq INF_k \leq Q_k \leq SUP_k \leq HI_k$
      - Anpassung von INF, Q, SUP bei Anforderung, Commit und Abort einer TA
    - Eigenschaften
      - Durchführung von Bereichstests bezüglich des Werteintervalls
      - Minimal-/Maximalwerte (LO, HI) dürfen nicht überschritten werden
      - hohe Parallelität ändernder Zugriffe möglich
    - Nachteile
      - spezielle Programmierschnittstelle
      - tatsächlicher Wert ggf. nicht abrufbar

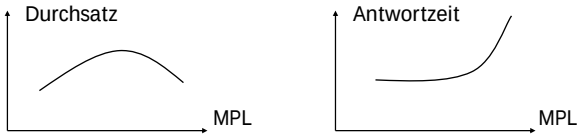
„High-Traffic'-Synchronisation (4)

- Escrow-Ansatz (Forts.)
  - aktueller Wert eines Escrow-Feldes (Forts.)
    - Beispiel: Zugriffe auf Feld mit LO = 0, HI = 500 (Anzahl freier Plätze)

T1	T2	T3	T4	INF	Q	SUP
				15	15	15
-5				10	10	15
	-8			2	2	15
		+4		2	6	19
			-3	-1		
c				2	6	14
		c		6	6	14
	a			14	14	14

Analyse (1)

- Synchronisation und Lastkontrolle
  - Lastkontrolle
    - „blinde" Durchsatzmaximierung
      - mehr aktive TA → mehr gesperrte Objekte → höhere Konflikt-WS → längere Sperrwartezeiten, höhere Deadlock-Raten → noch mehr aktive TA
    - Parallelitätsgrad (multi-programming level, MPL)
      - hat wesentlichen Einfluss auf das Leistungsverhalten, bestimmt Umfang der Konflikte bzw. Rücksetzungen
      - Gefahr von „Thrashing" bei Überschreitung eines kritischen MPL-Wertes



Analyse (2)

- Synchronisation und Lastkontrolle (Forts.)
  - Dynamische Lastkontrolle
    - „Statische“ MPL-Einstellung unzureichend
      - wechselnde Lastsituationen, mehrere Transaktionstypen
    - Idee:
      - dynamische Einstellung des MPL zur Vermeidung von „Thrashing“
    - möglicher Ansatz: Nutzung einer Konfliktrate bei Sperrverfahren
      - Konfliktrate = # gehaltener Sperren / # Sperren nicht-blockierter Transaktionen
      - kritischer Wert: ca. 1,3 (experimentell bestimmt)
      - Zulassung neuer TA nur, wenn kritischer Wert noch nicht erreicht ist
      - Bei Überschreiten erfolgt Abbrechen von TA

Weikum, G. et al.: The Comfort Automatic Tuning Project, in: Information Systems 19:5, 1994, 381-432

Zusammenfassung (1)

- Realisierung der Synchronisation durch Sperrverfahren
  - Sperren stellen während des laufenden Betriebs sicher, dass die resultierende Historie serialisierbar bleibt
  - bei einer Konfliktoperation blockieren sie den Zugriff auf das Objekt
  - es gibt mehrere Varianten (RX, RUX, ...)
  - Prädikatssperren verkörpern eine elegante Idee, sind aber in praktischen Fällen nicht direkt einsetzbar, ggf. Nutzung in der Form von Präzisionssperren
  - DBS-Standard: multiple Sperrgranulate durch hierarchische Sperrverfahren
  - Sperrverfahren sind pessimistisch und universell einsetzbar
  - Deadlock-Problem ist bei blockierenden Verfahren inhärent

Zusammenfassung (2)

- Einführung von Konsistenzebenen (vgl. DIS – SS03)
  - zwei (geringfügig) unterschiedliche Ansätze
    - basierend auf Sperrdauer für R und X
    - basierend auf zu tolerierenden Anomalien
  - Inkaufnahme von Anomalien reduziert die TA-Behinderungen
- Weitere Verfahren
  - RAX und RAC begrenzen Anzahl der Versionen und reduzieren Blockierungsdauern nur für bestimmte Situationen
  - Mehrversionen-Verfahren liefert hervorragende Werte bei der effektiven Parallelität und bei der Anzahl der Deadlocks, verlangt jedoch höheren Aufwand (Algorithmus, Speicherplatz)
  - reine OCC- und Zeitstempelverfahren erzeugen zu viele Rücksetzungen

Zusammenfassung (3)

- 'Harte' Synchronisationsprobleme
  - Beispiele
    - 'Hot Spots' / 'High Traffic'-Objekte
    - lange (Änderungs-) TA
  - Lösungen
    - wenn Vermeidungsstrategie nicht möglich ist, sind zumindest für Hochleistungssysteme Spezialprotokolle anzuwenden
    - Nutzung semantischen Wissens über Operationen / Objekte zur Reduzierung von Synchronisationskonflikten
  - allerdings
    - ggf. Erweiterung der Programmierschnittstelle
    - begrenzte Einsetzbarkeit
    - Zusatzaufwand
  - Dynamische Lastkontrolle erforderlich
    - Vermeidung von „Thrashing" bei wechselnden Lastsituationen, mehreren Transaktionstypen usw.
    - Nutzung einer Konfliktrate (1.3) zur dynamischen Einstellung der MPL