AG Datenbanken und Informationssysteme

Wintersemester 2006 / 2007

Prof. Dr.-Ing. Dr. h. c. Theo Härder
Fachbereich Informatik
Technische Universität Kaiserslautern

http://wwwdvs.informatik.uni-kl.de

8. Übungsblatt

Für die Übung am Donnerstag, **21. Dezember 2006**,
von 15:30 bis 17:00 Uhr in 13/222.

Aufgabe 1: Serialisierbarkeit von Historien

Gegeben sei folgende Historie:

H = <$r_3(c)$, $r_3(b)$, $r_4(b)$, $r_2(a)$, $r_4(a)$, $w_3(b)$, $w_2(a)$, $r_1(a)$, $w_1(a)$, $r_1(b)$, c_1, $r_5(c)$, $r_2(b)$, c_2, $w_5(c)$, c_5, c_3, c_4>

a) Beschreiben Sie allgemein das Ziel der Synchronisation.

b) Welche Probleme können in DB-Systemen auftreten, wenn keine Synchronisation vorgenommen wird. Geben Sie je ein Beispiel an.

c) Zeichnen Sie den Serialisierbarkeitsgraphen SG(H) und stellen Sie dann anhand dessen die Serialisierbarkeit von H fest.

d) Ist H auch kommutativitätsbasiert reduzierbar? Begründen Sie ihre Antwort. Ordnen Sie die Operationen entsprechend der Kommutativitätsregeln um.

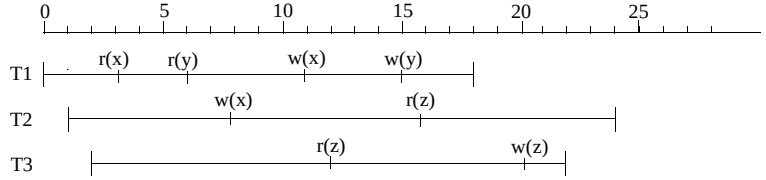
e) Ist H rücksetzbar? Wenn ja, dann begründen Sie Ihre Antwort. Ansonsten ordnen Sie die Operationen in H so um, dass die erzeugte Historie rücksetzbar wird. (Die Def. finden Sie ggf. in Ihrem alten GBIS Script)

f) Ermitteln Sie aus SG(H) alle möglichen seriellen Historien H_S^i.

g) Welche Einschränkungen der Klasse CSR kennen Sie? Beschreiben Sie diese mit ihren eigenen Worten.

Aufgabe 2: Vergleich verschiedener Scheduling-Algorithmen

Folgendes Bild zeigt den (hypothetischen) Ablauf dreier Transaktionen T1-T3, wenn sie ohne gegenseitige Beeinflussung (also kein Warten auf Sperren etc.) ablaufen könnten. Die zeitlichen Abstände zwischen den einzelnen Operationen **einer** Transaktion ergeben sich aus der "Denk"- oder Rechenzeit der Transaktionen. Diese Abstände verändern sich nicht, auch wenn eine Operation z.B. wegen eines Sperrkonfliktes für eine Weile blockiert wird oder nach einem Abort neu gestartet wird.



Ermitteln Sie basierend auf der so beschriebenen Eingabeschedule die Ausgabe, die folgende Scheduling-Algorithmen liefern würden:

h) SS2PL

i) TO (einfach, ein Zeitstempel)

j) BTO (getrennte Lese- und Schreibzeitstempel)

k) SGT

Gehen Sie davon aus, dass beim Sperrverfahren die (erfolgreiche, nicht blockierende) Sperranforderung und die Durchführung der Operation eine Aktion sind. Wird eine Transaktion durch den Scheduler abgebrochen, wird sie direkt neu gestartet.

Aufgabe 3: Optimistische Synchronisation

In dieser Aufgabe werden Validierungsregeln für optimistische Verfahren zur Mehrbenutzerkontrolle betrachtet. Folgende Abkürzungen werden verwendet:

RS_i = Read-Set (T_i) = Menge der Objekte, die Transaktion T_i liest

WS_i = Write-Set (T_i) = Menge der Objekte, die Transaktion T_i schreibt

a) Konstruieren Sie einen Schedule, der zeigt, dass die folgenden Regeln der Validierungsphase nicht notwendigerweise Serialisierbarkeit gewährleisten.
Regeln: Eine Transaktion T_v wird validiert, wenn für alle T_i mit $i < v$ eine der folgenden beiden Bedingungen erfüllt ist:

(1) T_i beendet die Schreibphase, bevor T_v die Lesephase beginnt.

(2) $WS_i \cap RS_v = \emptyset$ und T_i beendet die Lesephase, bevor T_v die Schreibphase beginnt.

b) Beweisen Sie, dass die unter a) aufgeführten Validierungsregeln Serialisierbarkeit garantieren, sofern für alle Transaktionen T_i gilt:
$WS_i \subseteq RS_i$, d. h. jedes Objekt, das geschrieben wird, muss zuvor gelesen werden.

c) Warum ist es ungünstig, Transaktionen in der Reihenfolge ihres Starts zu validieren? Welche andere Reihenfolge erlaubt eine schnellere Abfertigung der Transaktion, d. h. einen höheren Durchsatz?

Aufgabe 4: Sperrprotokolle verschiedener Konsistenzebenen

Mit LOCK (<name>, <modus>) wird für ein Objekt mit dem Namen <name> eine Sperre vom Typ <modus> angefordert. Durch UNLOCK <name> wird eine gewährte Sperre wieder zurückgegeben. Auf welchen Konsistenzebenen laufen die folgende Transaktionen? Welche der folgenden Protokolle sind zweiphasig in Bezug auf

a) Leser

b) Schreiber

c) Leser und Schreiber?

T_1	T_2	T_3	T_4	T_5
L_1:	LOCK (DB, IX)	LOCK (DB, IX)	LOCK (DB, IX)	LOCK (DB, IX)
READ NEXT IN R_1	LOCK (S_1, IX)	LOCK (S_1, IX)	LOCK (S_1, IX)	LOCK (S_1, IX)
IF !EOF GO TO L_1	LOCK (R_1, IX)	LOCK (R_1, IX)	LOCK (R_1, IX)	LOCK (R_1, IX)
LOCK (DB, X)	LOCK (t_{111}, R)	READ t_{111}	READ t_{111}	LOCK (t_{111}, R)
INSERT t_{11k}			READ t_{11i}	READ t_{111}
UNLOCK DB	READ t_{111}	READ t_{112}	UNLOCK R_1	
	UNLOCK t_{111}	READ t_{113}	LOCK (R_1, IX)	LOCK (t_{11i}, R)
		LOCK (t_{113}, X)	LOCK (t_{11i}, X)	READ t_{11i}
	LOCK (t_{11i}, X)	MODIFY t_{113}	MODIFY t_{11i}	LOCK (t_{11i}, X)
	MODIFY t_{11i}	READ t_{114}	UNLOCK t_{11i}	MODIFY t_{11i}
	LOCK (t_{11k}, R)		UNLOCK R_1	
	READ t_{11k}	LOCK (t_{11i}, X)	LOCK (R_1, R)	LOCK (t_{11k}, X)
	UNLOCK t_{11k}	MODIFY t_{11i}	READ t_{11k}	MODIFY t_{11k}
	LOCK (t_{11k}, X)		UNLOCK R_1	UNLOCK t_{11k}
	MODIFY t_{11k}	UNLOCK t_{11i}	LOCK (R_1, IX)	UNLOCK t_{11i}
	UNLOCK t_{11k}	UNLOCK t_{113}	LOCK (t_{11k}, X)	UNLOCK t_{111}
	UNLOCK t_{11i}	UNLOCK R_1	MODIFY t_{11k}	UNLOCK R_1
	UNLOCK R_1	UNLOCK S_1	UNLOCK t_{11k}	UNLOCK S_1
	UNLOCK S_1	UNLOCK DB	UNLOCK R_1	UNLOCK DB
	UNLOCK DB		UNLOCK S_1	
			UNLOCK DB	