Übungsblatt 3

Adrian Schollmeyer

Aufgabe 1

Sperrmodelle
- Unterscheidung zwischen Locks (Trennung von Transaktionen, verwaltet durch Sperrmanager) und Latches (Trennung von Threads, Zugriff auf Sperrtabelle, z. B. über Mutex realisiert)
- Sperrung Schreib- und Leseoperationen: $rl(x)$ (shared lock), $wl(x)$ (exclusive lock)
- Entsperrung: $ru(x)$, $wu(x)$ bzw. zusammengefasst $u(x)$

Sperrdisziplin Sämtliche Schreib- und Lesezugriffe erfordern, dass zunächst ein entsprechendes Lock gesetzt wurde (Lesen ist auch mit $wl(\cdot)$ möglich). Sperrungen sind nur für Objekte möglich, die nicht bereits von einem anderen Thread exklusiv gesperrt wurden Nach $rl(x)$ kann höchstens noch $wl(x)$ aquiriert werden. Nach $u(x)$ darf x von dieser Transaktion nie wieder gesperrt werden. Sämtliche Sperren müsse vor `commit` aufgehoben werden.

Sperrkompatibilitätsmatrix Die Sperrkompatibilitätsmatrix zeigt an, welche Sperren auf dasselbe Objekt zeitgleich in verschiedenen Transaktionen gesetzt werden können. Dies wird dadurch bestimmt, ob die zulässigen Aktionen nach der Sperre die Berechnungen der anderen Transaktionen beeinflussen können. Da Schreiboperationen dies immer tun, können im einfachen Modell keine anderen Sperren neben $wl(x)$ existieren (exclusive lock). Paralleles Lesen desselben Datensatzes führt jedoch zu keinen Problemen, weshalb Lesesperren parallel existieren dürfen.

Bei hierarchischen Sperren kommen zusätzlich intentionale Sperren hinzu. Dabei gilt weiterhin, dass Schreibsperren exklusiv sind. Allerdings können intentionale Lese- und Schreibsperren parallel existieren. Sobald jedoch stattdessen eine „echte“ Lese- oder Schreibsperre existiert, darf keine intentionale Sperre des anderen Typs existieren. Für echte Schreibsperren darf zudem auch keine andere intentionale Schreibsperre existieren.

Hierarchische Sperren In Baumstrukturen muss teilweise sicheregestellt werden, dass Sperren sich auch auf Kind- oder Elternknoten (teilweise) propagieren, da Baumoperationen manchmal nicht nur einzelne Knoten, sondern auch die darüber- oder darunterliegenden Knoten betreffen können.

Bei hierarchischen Sperren werden dabei folgende Regeln durchgesetzt:
- Sperren pflanzen sich nach unten fort.
- Sperren dürfen nicht von oben überschrieben werden

Hinzu kommen dabei neben den „echten“ Sperren noch sog. „intentionale“ Sperren, die Auskunft darüber geben, was für echte Sperren weiter unten in der Hierarchie existieren. Dadurch kann verhindert werden, dass potentiell inkompatible Operationen in verschiedenen Teilbäumen zu Operationen auf Elternknoten erfordern, die Konflikte verursachen könnten.

Beim hierarchischen Sperren wird immer auf dem Pfad von der Wurzel zum zu sperrenden Knoten jeder Knoten mit einer intentionalen Sperre belegt, die der echten Sperre des zu sperrenden Knotens entspricht. Falls nötig (z. B. weil eine Baumoperation dies erfordert), können intentionale Sperren auch verschärft werden (bspw. Umwandlung $iwl \rightarrow wl$, $irl \rightarrow rl$, $irl \rightarrow iwl$). Die Freigabe von Sperren erfolgt anschließend in der umgekehrten Reihenfolge, in der die Sperren angefordert wurden.

2PL Das Zwei-Phasen-Sperrprotokoll schreibt Regeln vor, zu welchen Zeitpunkten in der Transaktion Sperren gesetzt und wann sie freigegeben werden dürfen. Dabei wird die Transaktion in eine Anforderungs-, Nutzungs- und Freigabephase unterteilt, die in dieser Reihenfolge stattfinden. Nur während der Anforderungsphase dürfen Locks gesetzt werden. In der Nutzungsphase darf mit den Daten entsprechend der gesetzten Sperren gearbeitet werden. Anschließend werden in der Freigabephase die gesetzten Locks freigegeben.

Die Nutzungsphase ist dabei in die Anforderungs- und Freigabephase integriert. Der Übergang von der Anforderungs- in die Freigabephase findet dabei in dem Moment statt, wo die erste Sperre freigegeben wird. Es ist dabei nicht erforderlich, dass alle benötigten Sperren auf einmal gesetzt oder freigegeben werden!

Verschärft man das 2PC zu einem strikten 2-Phasen-Sperrprotokoll (S2PL), wird zusätzlich gefordert, dass die Freigabephase aus einer atomaren Freigabe aller Sperren auf einmal (bspw. am Ende der Transaktion) besteht. Dies hilft bei der Vermeidung kaskadierender Abbrüche.

Das konservative 2PL (C2PL) hingegen fordert das atomare Sperren aller benötigten Objekte zum Beginn der Transaktion. Dies vermeidet Deadlocks, erfordert aber auch, dass zu Beginn der Transaktion bekannt ist, welche Sperren benötigt werden.

Die Kombination aus C2PL und S2PL zu CS2PL ist auch möglich, sodass alle Sperren zu Beginn der Transaktion atomar gesetzt und am Ende der Transaktion atomar freigegeben werden müssen.

Baumprotokoll Das Baumprotokoll schreibt Regeln für das Setzen von Sperren in Baumstrukturen vor. Dies wird u. A. für Indexstrukturen genutzt, die bspw. mithilfe von B+-Bäume realisiert werden. Dabei gelten diese Regeln:

- Objekt o kann nur dann von T gesperrt werden, wenn sein direkter Vorgänger bereits von T gesperrt ist.
- Die erste Regel gilt nicht für die erste Sperre einer Transaktion.

- Sperren können jederzeit wieder freigegeben werden.
- Kein Objekt kann innerhalb einer Transaktion T zweimal gesperrt werden.

Man beachte hierbei insbesondere, dass Sperren jederzeit freigegeben werden können. Es ist nicht erforderlich, dass dabei eine bestimmte Reihenfolge eingehalten werden muss. Insbesondere erzwingt das Baumprotokoll kein 2PL.

Falls alle Transaktionen dem Baumprotokoll genügen, ist jeder korrekte Schedule serialisierbar.

MVCC MVCC vereint die Mehrversionen-Idee mit Synchronisation unter Nutzung diverser Synchronisationsverfahren wie Zeitstempelverfahren und Sperrverfahren. Bei Schreiboperationen in der Datenbank werden dabei alte Objekte zunächst nicht überschrieben, sondern es werden neue Versionen erzeugt. Leseoperationen in einer Transaktion sehen genau den Datenbestand zu dem Zeitpunkt, an dem die Transaktion erstellt wurde, außer natürlich die Transaktion hat das Objekt selbst verändert. Dies vermeidet Lesesperren gänzlich, wodurch Leseoperationen nicht warten müssen.

In MVCC kann ein Objekt nur geschrieben werden, wenn keine andere Transaktion vorher committet wurde, die das Objekt parallel auch geschrieben hat. Bei solchen Konflikten gilt „First Committer Wins“, d. h. die erste Transaktion, die das Objekt schreibt, wird ausgeführt, während andere in Konflikt stehende Transaktionen abgebrochen werden. Entsprechend muss eine Transaktion zum Commit-Zeitpunkt exklusive Sperren auf den geänderten Objekten haben.

Zur Realisierung von MVCC werden zu allen Tupeln Metadaten hinzugefügt: Eine optional gesetzte Transaktions-ID fungiert dabei als Schreibsperre (gesperrt, falls gesetzt; entsperrt, fals nicht gesetzt), ein Start-Zeitstempel gibt den Zeitpunkt an, ab dem ein Tupel gültig ist (anfangs 0, bei Löschung auf ∞), ein End-Zeitstempel gibt den Zeitpunkt an, ab dem die Gültigkeit des Tupels endet (anfangs 0) und optional wird noch ein Verweis auf das Tupel der nächsten Version eines Objekts angegeben.

Aufgabe 2

(a)

Schedule: $s = T_0 rl_1(x) r_1(x) u_1(x) wl_2(x) w_2(x) rl_2(y) r_2(y) wl_3(x) w_3(x) u_3(x) rl_2(z) r_2(z) u_2(x) u_2(y) u_2(z) \in CSR$, aber Konflikt zwischen $wl_2(x)$ und $wl_3(x)$. Da jedoch wegen 2PC T_2 noch nicht entsperren kann (da später noch Lesezugriffe auf z stattfinden sollen), wird ein solcher Schedule nicht akzeptiert.

(b)

Angenommen 2PC würde $s \notin CSR$ akzeptieren. Dann hat s einen Zyklus im Konfliktgraphen aus Transaktionen $T_1, \ldots, T_n$, o. B. d. A. $n = 2$. Dann $p_1(x) \rightarrow_s p_2(x)$ mit

$p_1(x) = w_1(x)$ oder $p_2(x) = w_2(x)$, also muss T_1 oder T_2 $wl_i(x)$ haben. Außerdem $q_2(x) \rightarrow_s q_1(x)$ analog, also muss T_1 oder T_2 $wl_j(x), j \neq i$ haben.

Da beide Transaktionen auf Objekte zugreifen, die von der jeweils anderen Transaktion gesperrt worden sind (und das nicht nacheinander, da sonst kein Zyklus entstünde), müssen sie gleichzeitig Sperren auf die gleichen Objekte halten, was jedoch nicht möglich ist. Alternativ müssten Sperren freigegeben werden, wonach jedoch gemäß 2PL keine weiteren Sperren mehr von der jeweiligen Transaktion angefordert werden dürften. Somit kann die Transaktion keine Sperre für das zweite Objekt anfordern und es damit nicht bearbeiten, weshalb es keinen Zyklus geben kann. Widerspruch, also akzeptiert 2PL nur Schedules $s \in CSR$. □

Aufgabe 3

(a)

- $s \in CSR$ (Konfliktgraph hat $T_1 \rightarrow T_3$, $T_1 \rightarrow T_2$, $T_3 \rightarrow T_2$)
- $s \in RC$ (RaW $w_1(x) \rightarrow_s r_2(x) \rightarrow_s r_3(x)$ und $c_1 \rightarrow_s c_2 \rightarrow_s c_3$)
- $s \notin ACA$ (RaW $w_1(x) \rightarrow_s r_2(x)$, aber nicht $c_1 \rightarrow_s r_2(x)$)

(b)

2PL:

- Anforderungsphase T_1 von Anfang bis $w_1(x)$; $wu_1(x)$ muss vor $r_2(x)$ kommen, startet Freigabephase, $ru_2(x)$ danach irgendwann vor c_1.
- Anforderungsphase T_2 von Anfang bis $w_2(z)$, Freigabephase in c_2.
- Anforderungsphase T_3 von Anfang bis $w_3(z)$, Freigabephase direkt nach $w_3(z)$ mit $u_3(z)$, Rest beliebig bis c_3.

S2PL:

- Anforderungsphase jeweils wie 2PL
- T_2, T_3 warten vor $r_2(x)$, $r_3(x)$ auf T_1, bis c_1
- T_2 wartet vor $w_2(z)$ auf T_3 bis c_3

C2PL:

- T_1 startet, fordert $wl_1(x)$ atomar an
- T_2 wartet auf c_1
- T_3 wartet auf c_1

- c_1
- T_3 fordert $wl_3(z), rl_3(x), rl_3(z)$ atomar an
- T_2 wartet auf T_3 bis $wu_3(z)$
- $wu_3(z)$
- T_2 fordert atmoar $rl_2(y)$, $rl_2(x)$, $wl_2(z)$ an
- Rest beliebig

TO Seien $ts(T_1) = 1$, $ts(T_2) = 2$, $ts(T_3) = 3$.

	mrs_x	mws_x	mrs_y	mws_y	mrs_z	mws_z
	0	0	0	0	0	0
$r_1(x)$	1					
$r_2(y)$			2			
$w_1(x)$		1				
$r_3(z)$					3	
$r_2(x)$	2					
$r_3(x)$	3					
$w_3(z)$						3
$w_2(z)$					`reject`	`reject`

2V2PL

T_1	T_2	T_3
$rl_1(x)$		
$r_1(x)$		
	$rl_2(y)$	
$wl_1(x)$		
$w_1(x_1)$		
		$rl_3(z)$
		$r_3(z)$
	$rl_2(x)$	
	$r_2(x)$	
		$rl_3(x)$
		$r_3(x)$
		$wl_3(z)$
		$w_3(z_3)$
	$wl_2(z) \Downarrow$	
$cl_1(x)$		
c_1		
		$cl_3(z)$
		c_3
	$wl_2(z)$	
	$w_2(z_2)$	
	$cl_2(z)$	
	c_2	

Aufgabe 4

Aufgabe 5

(a)

Optimistische Synchronisationsverfahren arbeiten auf der Grundannahme, dass es nur wenige tatsächliche Konflikte zwischen Datenbankoperationen gibt und dass eine Konfliktvermeidung durch Sperren vor Datenzugriffen kostenintensiver ist als eine nachträgliche Konfliktbehandlung. Optimistische Verfahren zeichnen sich vor allem dadurch aus, dass Anfragen grundsätzlich erst einmal parallel und ohne Konfliktprüfung durchgeführt werden können. Dabei wird durch Shadow Paging verhindert, dass noch nicht committete Transaktionen den Datenbestand ändern, mit dem andere Transaktionen arbeiten.

Für die anschließende Konfliktprüfung benötigen optimistische Verfahren zusätzliche Erfassungen, auf welche Objekte durch welche Transaktionen wie zugegriffen wurde (d. h. Read Sets und Write Sets). Bei der Konfliktprüfung muss dann durch Vergleich dieser Mengen geprüft werden, ob es zu irgendwelchen Konflikten kommt. Im Konfliktfall müssen dann einzelne Transaktionen abgebrochen werden, wodurch die Arbeit, die durch diese Transaktionen erledigt wurde, rückgängig gemacht wird.

(b)

$$ts_A(T_i) < ts_V(T_j) \wedge ts_A(T_j) < ts_V(T_i) \wedge \\ (RS(T_i) \cap WS(T_j) \neq \varnothing \vee RS(T_j) \cap WS(T_i) \neq \varnothing)$$

(c)

Die Read- und Write-Sets einer Transaktion beinhalten die von der Transaktion gelesenen respektive geschriebenen Objekte. Diese werden bei zentralisierten Architekturen im Koordinator, bei dezentralen Architekturen partiell in den beteiligten Objektmanagern geführt. In letzterem Fall wird die Schnittmengenbildung verteilt ausgeführt. Bei bereits einem Element in einer der Schnittmengen ist ein zu behandelnder Konflikt erkannt worden und eine der Transaktionen muss abgebrochen werden.