Übungsblatt 1

Adrian Schollmeyer

Aufgabe 1

(a)

- Subjekt
  - führen Anfragen aus, erstellen und beenden Transaktionen
  - reden mit Transaktionsmanager
  - bspw. Clients einer Anwendung
- Transaktionsmanager
  - Schnittstelle zwischen Subjekt und Objektmanagern
  - verwaltet Transaktionen
  - sichert ACID-Eigenschaften bei zeitgleicher paralleler Nutzung und Ausfällen
  - bspw. Transaktionsmanager in DBMS
- Objektmanager
  - Schnittstelle für Transaktionsmanager, um auf Objekte zuzugreifen
  - Koordinieren Lese- und Schreiboperationen auf Objekten
  - bspw. DBMS, Webserver, Speicher

Implementierung:

- atomare Instruktionen in der CPU (z. B. Intel TSX)
- Instruktionen auf Betriebssystemebene (z. B. für SSDs)
- Journaling in Dateisystemen (z. B. ext3)
- Programmiersprachen (z. B. Transaktionsoperationen in C++11)

(b)

- Zweck:
  - Sicherung der ACID-Eigenschaften bei parallelen Zugriffen und Ausfällen
  - Vermeidung von Locking bei parallelen Zugriffen
  - Atomarität von Folgen von Operationen
- Verschachtelte Transaktionen:

  - Vorteile: Fehler in Untertransaktionen können behandelt werden, ohne dass die gesamte Transaktion fehlschlägt, aber trotzdem Atomarität über die gesamte Transaktion; Parallelität über Untertransaktionen möglich
  - Kosten: Komplexität der Parallelität, zusätzliche Regelprüfungen bei Sicherung
  - eignet sich z. B. bei vielen lokalen Teilnehmern
- Verteilte Transaktionen:
  - Vorteile: Objektmanager können nach Bedarf hinzugezogen werden, Interaktion zwischen verschiedenen Objektmanagern möglich, Lastverteilung bei der Koordination von Transaktionen, Eliminierung eines sPof (TM) durch dezentrales TM
  - Anwendungsfälle: Transaktionen, die sich über mehrere Systeme (bspw. Buchungssysteme) erstrecken, replizierte Datenbanken
  - Kosten: OM müssen Koordinatorrolle übernehmen können, höhere Softwarekomplexität

(c)

Nur beim ersten OM wird `BeginTransaction` aufgerufen. Dieser wird daraufhin Koordinator. Auf anderen OM wird `JoinTransaction` aufgerufen und diese nehmen dann nur an der Transaktion des ersten OM teil.

(d)

Zentrales TM:

TM TM ist sPof, daher ist der Ausfall maximal kritisch. Bei Ausfall des TM können keine Transaktionen mehr ausgeführt werden und bestehende, nicht abgeschlossene Transaktionen gehen möglicherweise verloren.

OM Ausfall sorgt für Abbruch aller Transaktionen, die Objekte des OM benutzen. Transaktionen, die diesen OM nicht benutzen, können erfolgreich abgeschlossen werden.

Verteiltes TM:

- Da erster OM als TM agiert, ist dieser sPof für alle Transaktionen, die über ihn koordiniert werden. Der Ausfall dieses OM betrifft alle Transaktionen, die über den OM koordiniert werden, sowie alle Transaktionen, an denen der OM lediglich teilnimmt.

  Kritikalität des ersten OM gleich wie zentrales TM, da der erste OM auch in zentralem TM für alle Transaktionen, für die er sPof ist, angesprochen worden wäre und dort ein Ausfall auch zum Abbruch dieser Transaktionen geführt hätte.

- Andere OMs gleich wie in zentralem TM.

Aufgabe 2

(a)

- `abortTransaction` an Punkt 1:
  - T_{11} wird abgebrochen (O_{11} wird resettet)
  - T_1 wird abgebrochen, da Untertransaktion abgebrochen wurde (O_1 wird resettet)
  - T_2 und Untertransaktionen werden ausgeführt
  - T_{top} wird abgebrochen, da T_1 abgebrochen wurde, daher werden auch T_2 und Untertransaktionen abgebrochen (O_2 wird resettet)
  - keine Änderungen werden gespeichert
- ausfallbedingter Abbruch von T_2 an Punkt 2:
  - T_{21} und T_{211} werden abgebrochen
  - T_{22} wird abgebrochen
  - O_2 wird zurückgesetzt (Änderungen von O_{21}, O_{22} dürfen auch nicht geschrieben werden!)
  - T_{top} wird abgebrochen, daher auch T_1 und Untertransaktionen
  - keine Änderungen werden gespeichert
- `endTransaction` an Punkt 3:
  - alle Untertransaktionen sind commitet
  - alle Änderungen an den Objekten werden rekursiv gespeichert

Zu 1. und 2. Szenario: die Elterntransaktionen können im Fehlerfall prinzipiell auch entscheiden, trotzdem weiterzumachen. In dem Fall können die Änderungen der Objekte, deren Transaktionen nicht abgebrochen wurden, geschrieben werden.

(b)

- T_1 und T_2
- T_{21} und T_{22}

Alle, Transkationen, die keine Eltern oder Kind sind, können nebenläufig ausgeführt werden.

T_1 und T_2 auf versch. Server. Wo synchronisieren?

- T_{21} mit T_{22}
- T_{211} mit T_{22}

(c)

Aufgabe 3

(a) ACID:

Atomarität Die Operation wird entweder vollständig oder gar nicht ausgeführt.

Konsistenz Zwischenergebnisse werden nur in lokalen Kopien geschrieben und sind nicht von außerhalb der Transaktion einsehbar.

Isolation Objekte, der Transaktion sind gesperrt, sodass andere Transaktionen nicht darauf zugreifen können.

Dauerhaftigkeit Nach erfolgreichem Ende einer Transaktion sind alle Änderungen auf persistenten Speicher geschrieben.

(b) Transaktion

- Folge von Operationen, die isoliert ausgeführt werden und bei deren Ausführung die ACID-Eigenschaften garantiert werden.
- Folge von Operationen, die entweder ganz oder gar nicht ausgeführt werden.

(c) Schedule

- Reihenfolge der Ausführung von Operationen verschiedener Transaktionen
- vollständiger Schedule, wenn alle Operationen der beteilten Transaktionen (inkl. commit/abort)
- „normaler“ Schedule ist Prefix eines vollständigen Schedules

(d) Serieller Schedule

- Schedule, bei dem alle Operationen einer Transaktionen hintereinander ausgeführt werden
- Operationen mehrerer Transaktionen werden nicht verschränkt ausgeführt

(e) $RF(s)$

- Menge von Lesezugriffen, bei denen eine Transaktion auf Daten einer anderen zugreift
- $RF(s) := \{(T_i, x, T_j) \mid r_j(x) \text{ liest von } T_i\}$

(f) Sichtserialisierbarkeit

- Schedule ist sichtäquivalent zu einem seriellen Schedule
- sichtäquivalent, wenn:

$$op(s) = op(s') \tag{1}$$
$$RF(s) = RF(s') \tag{2}$$
$$\tag{3}$$

(g) Konfliktrelation

- $C(s) := \{(p, q) \mid p, q \text{ sind in Konflikt } \wedge p \rightarrow_s q\}$
- Menge aller Operationen, die Konflikte erzeugen (d. h. bei denen die Reihenfolge Einfluss auf das Ergebnis hat)
- $\text{conf}(s) := C(s) \setminus \{(p, q) \mid (p \in t' \vee q \in t') \wedge t' \in \text{aborted}(s)\}$ (Menge aller Konflikte, deren beteiligte Operationen nicht Teil einer abgebrochenen Transaktion sind)

(h) Konfliktserialisierbarkeit

- $op(s) = op(s')$
- $\text{conf}(s) = \text{conf}(s')$
- konfliktäquivalent zu seriellem Schedule
- gleiche Konflikte (der nicht abgebrochenen Transaktionen) wie bei einem seriellen Schedule

Konfliktarten:

Lost Update Zwei Transaktionen schreiben auf das gleiche Objekt und die erste Transaktion geht verloren.

Dirty Read Es wird etwas gelesen, was später überschrieben wird.

Non Repeatable Read Wenn ich etwas zweimal lese und es zwischendurch Änderungen gab, hat es beim zweiten Mal einen anderen Wert.

Aufgabe 4

(a) keine (wird ja nur gelesen)

(b)
- T_1 kann mit falschem Wert von A weiterarbeiten, der von T_2 geschrieben wurde, falls T_2 abgebrochen wird (*Dirty Read*).

(c)
- *Non-Repeatable Read*, wenn man in T_1 nochmal r_A ausführt

(d)
- Abbruch von T_1 nachdem T_2 geschrieben wurde, könnte fälschlicherweise Wert von A zurücksetzen (*Lost Update*)

Aufgabe 5

(a)

Sichtserialisierbarkeit:

- $RF(s)$ aufstellen: Operationen finden, die ein Objekt schreiben. Operationen finden, die danach von dem Objekt lesen.
- $RF(s)$ muss für *alle* seriellen Schedules aufgebaut werden
- Man braucht am Anfang eine Initialisierungstransaktion T_0, die alle Objekte schreibt, sowie eine Terminierungstransaktion T_∞, die alle Objekte liest.

Konfliktserialisierbarkeit:

- $C(s)$ aufstellen (eigentlich conf(s) und für alle seriellen Schedules
- paare von Operationen finden, wo ein Lese-/Schreibzugriff in einer Transaktion auf einen Schreibzugriff in einer anderen Transaktion folgt

$$
\begin{aligned}
RF(s_1) &= \{(T_0, x, T_1), (T_0, z, T_3), (T_1, x, T_2), (T_0, x, T_2), (T_3, x, T_\infty), (T_0, z, T_\infty)\} \\
RF(T_1\,T_2\,T_3) &= \{(T_0, x, T_1), (T_0, z, T_2), (T_1, x, T_2), (T_0, z, T_2), (T_3, x, T_\infty), (T_0, z, T_\infty)\} \\
RF(T_1\,T_3\,T_2) &= \{(T_0, x, T_1), (T_0, z, T_1), (T_0, z, T_3), (T_3, x, T_2), (T_3, x, T_\infty), (T_0, z, T_\infty)\} \\
RF(T_2\,T_1\,T_3) &= \\
RF(T_2\,T_3\,T_1) &= \\
RF(T_3\,T_1\,T_2) &= \\
RF(T_3\,T_2\,T_1) &= \\
C(T_1\,T_2\,T_3) &= \{(r_1(x)\,w_3(x)), (w_1(x)\,r_2(x)), (w_1(x)\,w_3(x)), (r_2(x)\,w_3(x))\} \\
C(s) &= C(T_1\,T_2\,T_3) \implies s_1 \in \text{CSR}
\end{aligned}
$$

s_1 sichtserialisierbar (kriegt man u. A. auch dadurch, dass es in CSR ist) und konfliktserialisierbar.
$s_2 \notin$ VSR, daher auch $s_2 \notin$ CSR.
$s_3 \in$ VSR und $s_3 \in$ CSR.
$s_4 \in$ VSR und $s_4 \in$ CSR.

(hier eigene Vorbereitung, nicht ganz korrekt)

$$RF(s_1) = \{(T_1, x, T_2)\}$$
$$RF(s_2) = \{(T_2, y, T_1), (T_1, x, T_2)\}$$
$$RF(s_3) = \{\}$$
$$RF(s_4) = \{(T_2, x, T_1), (T_2, y, T_1)\}$$
$$C(s_1) = \{(w_1(x), r_2(x)), (r_1(x), w_3(x)), (w_1(x), w_3(x))\}$$
$$C(s_2) = \{(w_2(y), r_1(y)), (w_1(x), r_2(x))\}$$
$$C(s_3) = \{(r_1(x), w_2(x)), (w_1(x), w_2(x)), (r_1(x), w_3(x)), (w_1(x), w_3(x)), (w_2(x), w_3(x))\}$$
$$C(s_4) = \{(w_2(x), r_1(x)), (w_2(y), r_1(y)), (w_2(y), w_1(y))\}$$

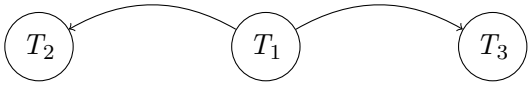
Serielle Schedules:

$$s_1' = r_1(x)\, w_1(x)\, r_1(z)\, r_2(x)\, r_3(z)\, w_3(x)$$
$$s_2' = r_1(x)\, w_1(x)\, r_1(y)\, r_2(y)\, w_2(y)\, r_2(x)$$
$$s_3' = r_1(x)\, w_1(x)\, r_2(y)\, w_2(x)\, r_3(y)\, w_3(x)$$
$$s_4' = r_1(x)\, r_1(y)\, w_1(y)\, w_2(x)\, w_2(y)\, w_3(z)$$
$$RF(s_1') = \{(T_1, x, T_2)\}$$
$$RF(s_2') = \{\}$$
$$RF(s_3') = \{(T_2, x, T_3)\}$$
$$RF(s_4') = \{\}$$

Sichtserialisierbarkeit:

- s_1 : ✓
- s_2 : ×
- s_3 : ×
- s_4 : ×

Konfliktserialisierbarkeit s_1:



Daher s_1 konfliktserialisierbar. Die anderen Schedules sind nicht konfliktserialisierbar, da sie nicht sichtserialisierbar sind.

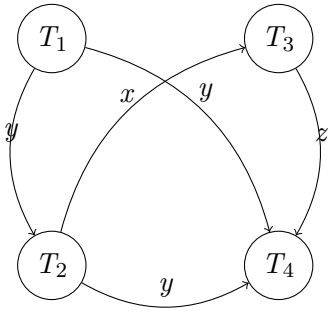
(b)

$$C(s) = \{(r_2(x), r_3(x)), \dots\}$$

Konflikte zwischen:

- (T_1, T_2)
- (T_3, T_4)

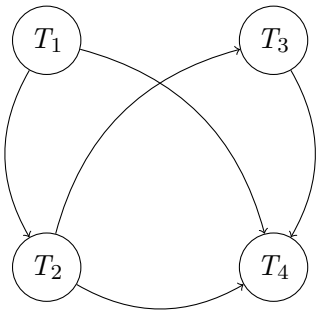
- (T_2, T_3)
- (T_2, T_4)
- (T_1, T_4)



Da der Graph kreisfrei ist, ist $s \in \text{CSR}$.

(ab hier eigene Vorbereitung, falsch)

$$C(s) = \{(w_3(z), r_4(z)), (r_2(x), w_3(x)), (w_1(y), w_2(y)), (w_1(y), r_4(y)), (w_2(y), r_4(y))\}$$



Der Graph ist kreisfrei. Daher ist s konfliktserialisierbar.

Serieller Schedule:

$$s' = r_1(y)\, w_1(y)\, r_2(y)\, w_2(y)\, r_3(x)\, w_3(z)\, w_3(x)\, r_4(z)\, r_4(y)$$