Nachtrag Übungsserie 7

(Transaktionen, geändert am 06.02.08)

Serialisierbarkeit und Konfliktserialisierbarkeit

nachzulesen in: Garcia-Molina, Ullman, Widom. Database Systems – The Complete Book. 917 – 932.

- *Transaktion*: geordnete Folge von Aktionen (z. B. read, write)
- *Schedule*: geordnete Folge von Aktionen mehrerer Transaktionen
- *serieller Schedule*: Schedule, in dem keine Verschachtelung von Transaktionen auftritt, sondern erst eine Transaktion komplett abgearbeitet wird, bevor die Abarbeitung der nächsten Transaktion beginnt
- *serialisierbarer Schedule*: Schedule S, der *äquivalent* zu einem seriellen Schedule S' ist, d. h. Effekt von S bezogen auf einen beliebigen Datenbankzustand stimmt mit dem Effekt von S' überein
- *konfliktserialisierbarer Schedule*: Schedule S, der *konfliktäquivalent* zu einem seriellen Schedule S' ist, d. h. S kann durch eine Folge von Vertauschungen von nicht-konfligierenden Aktionen in S' überführt werden
- *konfligierende Aktionen $\alpha_i(O)$ und $\alpha_j(O)$*: α_i und α_j sind zwei Aktionen (Schreib- oder Leseoperation) unterschiedlicher Transaktionen ($i \neq j$) und mindestens eine der beiden Operationen ist eine Schreiboperation (beachte, dass beide Aktionen auf dem gleichen Datenbankobjekt O arbeiten)

Es gilt: Ist S konfliktserialisierbar, so ist S auch serialisierbar. In diesem Sinne ist die Konfliktserialisierbarkeit eine *hinreichende* Bedingung für die Serialisierbarkeit. Die Konfliktserialisierbarkeit ist aber keine *notwendige* Bedingung für die Serialisierbarkeit, da serialisierbare Schedules existieren, die nicht konfliktserialisierbar sind.

Offensichtlich ist jeder serielle Schedule sowohl serialisierbar als auch konfliktserialisierbar.

Ferner gilt: Der Konfliktgraph G_S ist genau dann azyklisch, wenn der zugehörige Schedule S konfliktserialisierbar ist. Unter Ausnutzung des obigen Zusammenhangs folgt darüber hinaus: Ist G_S azyklisch, so ist S auch serialisierbar. Die Umkehrung dieser Aussage gilt aber i. A. nicht.

Bemerkungen zur Serialisierbarkeit (aus der Übung vom 06.02.08)

Es ist zu entscheiden, ob der Schedule

$$S = \langle r_3(Y), w_1(X), w_2(X), w_2(Y), w_1(Y), r_2(X), r_3(Y), w_3(Y) \rangle$$

serialisierbar ist. Dazu müssen wir prüfen, ob S äquivalent zu einem seriellen Schedule (bzgl. der drei Transaktionen T_1, T_2 und T_3 ist).

Es ist offensichtlich, dass nur die Äquivalenz bezüglich des seriellen Schedules $S' = T_1, T_2, T_3$ überprüft werden muss (T_3 muss zuletzt kommen, da w_3 die letzte Schreiboperation auf Y ist; T_2 muss nach T_1 kommen, da w_2 die letzte Schreiboperation auf X ist).

Demzufolge müssen wir die Äquivalenz bezüglich

$$S' = \langle w_1(X), w_1(Y), w_2(X), w_2(Y), r_2(X), r_3(Y), r_3(Y), w_3(Y) \rangle$$

überprüfen.

Betrachten wir die letzte Schreiboperation auf Y ($w_3(Y)$) in S bzw. S'. Wir müssen (schlimmstenfalls) davon ausgehen, dass der Wert, den T_3 auf Y schreibt, von allen Werten abhängig ist, die T_3 zuvor aus der Datenbank gelesen hat.

Im Schedule S liest T_3 zwei Werte: den Wert von Y (z. B. 10) bevor eine der drei Transaktionen auf Y schreibt und den Wert von Y (z. B. 20) nachdem T_1 und T_2 auf Y geschrieben haben. Wir nehmen an, dass T_3 am Ende die Summe der beiden zuvor gelesenen Werte in Y schreibt (in diesem Fall $10 + 20 = 30$).

Im Schedule S' liest T_3 ebenfalls zweimal den Wert von Y, aber in beiden Fälle nachdem T_1 und T_2 den Wert von Y verändert haben (könnten). Bezogen auf das Beispiel liest T_3 zweimal den Wert 20 und schreibt somit schließlich $20 + 20 = 40$ in Y.

Offensichtlich sind die sich durch die Ausführung von S bzw. S' ergebenen Datenbank-Zustände *nicht* identisch (wird S ausgeführt, so hat Y den Wert 30; wird S' ausgeführt, so hat Y den Wert 40). Daher ist der Schedule S *nicht serialisierbar*.

Sperren (*locks*)

nachzulesen in: Garcia-Molina, Ullman, Widom. Database Systems – The Complete Book. 932 – 951.

- Lesesperre (*shared-, S-lock*) auf Datenbankobjekt O anfordern: $sl_i(O)$ (i entspricht der Transaktionsnummer)

- Schreibsperre (*exclusive-, X-lock*) auf O anfordern: $xl_i(O)$
- alle Sperren auf O freigeben: $u_i(O)$
- *Konsistenz einer Transaktion*:
  - vor der Ausführung einer Schreiboperation auf ein Datenbankobjekt muss eine Schreibsperre angefordert werden
  - vor der Ausführung einer Leseoperation auf ein Datenbankobjekt muss eine Lese- oder Schreibsperre angefordert werden
  - am Ende der Transaktionen müssen alle Sperren freigegeben worden sein
- *2PL-Transaktion*: nach der ersten *unlock*-Operation dürfen keine Sperren (Lese- und Schreibsperren) mehr angefordert werden (für kein Datenbankobjekt)
- *Legaler Schedule*: ein Datenbankobjekt kann zu einem Zeitpunkt entweder nur von genau einer Transaktion exklusiv (mittels einer Schreibsperre) gesperrt sein oder von mehreren Transaktionen mit Lesesperren gesperrt sein, d. h.
  - zwischen einem $xl_i(O)$ und dem ersten danach auftretenden $u_i(O)$ darf kein $xl_j(O)$ und $sl_j(O)$ für $j \neq i$ stehen
  - zwischen einem $sl_i(O)$ und dem ersten danach auftretenden $u_i(O)$ darf kein $xl_j(O)$ für $j \neq i$ stehen

Es gilt: Jeder legale Schedule S von konsistenten, 2PL-Transaktionen ist konfliktserialisierbar. In diesem Fall erhält man den konfliktäquivalenten, seriellen Schedule, indem man für jede Transaktion T_i die erste *unlock*-Operation u_i^* in S betrachtet und die Transaktionen schließlich entsprechend der Reihenfolge der u_i^* in S ordnet.