TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Dr. Thomas Neumann

Übung zur Vorlesung
***Einsatz und Realisierung von Datenbanksystemen* im SoSe15**
Moritz Kaufmann (moritz.kaufmann@tum.de)
http://db.in.tum.de/teaching/ss15/impldb/

Blatt Nr. 4

Hausaufgabe 1 Nennen Sie die Vorteile und Nachteile von Deadlockerkennung / Vermeidung durch:

- Timeouts
- Wartegraphen
- Preclaiming
- Zeitstempel

Sind Kombinationen denkbar/sinnvoll?

- Timeouts
  - + Verfahren ist einfach und billig
  - - Deadlocks werden erst mit Verzögerung erkannt
  - - Es gibt false positives
- Wartegraphen
  - + Es werden echte Deadlocks schnell erkannt
  - - Hohe kosten
- Preclaiming
  - + Deadlocks treten nicht auf
  - - Verringert Parallelität
- Zeitstempel
  - + Deadlocks treten nicht auf
  - - Viele Transaktionen müssen zurückgesetzt werden, obwohl nie ein Deadlock auftreten würde – false positives.

Beispielsweise kann Timeout und Wartegraph kombiniert werden. Hierbei wird der Wartegraph erst erzeugt, wenn ein Timeout auftritt. Es treten so im Vergleich zum Timeout-Verfahren keine false positives mehr auf und das Verfahren bleibt im Regelfall billig, da der Wartegraph „on demand“ erzeugt wird. Das Problem der verzögerten Erkennung von Deadlocks bleibt allerdings bestehen.

Hausaufgabe 2 Weisen Sie (halbwegs) formal nach, dass das 2PL-Protokoll nur serialisierbare Historien zulässt.

Um zu zeigen, dass H serialisierbar ist, weisen wir nach, dass der zugehörige Serialisierbarkeitsgraph $SG(H)$ azyklisch ist:

1. Falls in $SG(H)$ die Kante $T_i \to T_j$ auftritt, so steht eine Operation von T_i mit einer Operation T_j bezüglich eines Datums A in Konflikt. Der Vorüberlegung 1.2 zufolge muss T_i dann die Sperre auf A wieder freigegeben haben, ehe T_j die Sperre darauf erhält.

2. Nehmen wir an, dass der Pfad $T_i \to T_j \to T_k$ in $SG(H)$ auftritt. Aus Schritt 2.1 wissen wir, dass T_i also eine Sperre freigeben musste, ehe T_j eine Sperre erhielt. Analog gab T_j zuvor eine Sperre frei, ehe T_k eine Sperre erlangt. Da wir wissen, dass eine Transaktion keine Sperren mehr anfordert, nachdem sie bereits andere Sperren wieder freigegeben hat, können wir per Transitivität schlussfolgern, dass T_i eine Sperre freigeben muss, ehe T_k eine Sperre erhalten kann. Per Induktion kann man dann folgern, dass, wenn in $SG(H)$ der Pfad $T_1 \to T_2 \to \ldots T_n$ auftritt, T_1 zuerst eine Sperre freigeben muss, ehe T_n eine Sperre erhält.

   Anzumerken ist, dass sich diese Sperren nicht immer auf dasselbe Datum beziehen müssen. T_i und T_j stehen im Allgemeinen bezüglich eines anderen Datums in Abhängigkeit als T_j und T_k. Beispielsweise kann die Historie

   $$r_i(A) \to w_j(A) \to w_j(B) \to r_k(B)$$

   zum Pfad $T_i \to T_j \to T_k$ in $SG(H)$ führen.

3. Damit ist es ein Leichtes, zu zeigen, dass $SG(H)$ azyklisch ist. Nehmen wir an, in $SG(H)$ tritt der Zyklus $T_1 \to T_2 \to \ldots T_n \to T_1$ auf. Nach Schritt 2.2 muss dann T_1 eine Sperre freigegeben haben, ehe es eine andere Sperre erhält. Dies widerspricht jedoch Eigenschaft (4) des 2PL-Protokolls, bzw. 1.3. Da $SG(H)$ zyklenfrei ist, können wir nach dem Serialisierbarkeitstheorem folgern, dass H serialisierbar ist.

Hausaufgabe 3 Beim „multiple-granularity locking“ (MGL) werden Sperren von oben nach unten (top-down) in der Datenhierarchie erworben. Zeigen Sie mögliche Fehlerzustände, die eintreten könnten, wenn man die Sperren in umgekehrter Reihenfolge (also bottom-up) setzen würde.

Siehe Übungsbuch

Gruppenaufgabe 4 (Lösung während der Übung) You are using a database management system that implements the ARIES protocol for logging and recovery. The system uses strict two-phase locking, and the “no-force” and steal strategies. The database has just two items in it, X with starting value 10, and Y with starting value 100. You start three transactions at the same time (TA, TB, and TC):

```
TA:
  BEGIN TRANSACTION
  X = X + 1
  Y = Y * 3
  COMMIT TRANSACTION
```

```
TB:
  BEGIN TRANSACTION
  Y = Y * 2
  X = X + 5
  COMMIT TRANSACTION

TC:
  BEGIN TRANSACTION
  X = X * 10
  COMMIT TRANSACTION
```

These three transactions are the only activity in the system. The system crashes due to a power failure soon after you start the transactions. You are not sure whether or not any of them completed. You look at the disk while the system is down and see that, Y has the value 200. You restart the system and let the database recovery procedure complete. You query the database for the value of X, and it returns the value 110.

Please write down a log, as it would have appeared on the disk while the system was down, that is compatible with the above story. You only need to include Update (U), Commit (C), and Abort (A) records. Note that the database system described above (ARIES, 2PL) may abort a transaction. Specify the transaction (TA, TB, or TC) and record type (U/C/A) for each record. For Updates specify additionally the item being written (X or Y) and the new value being written.

TID (TA,TB,TC)	Type (U/C/A)	Item (X,Y)	New Value

TID (TA,TB,TC)	Type (U/C/A)	Item (X,Y)	New Value
TA	U	X	11
TB	U	Y	200
TB	A		
TA	U	Y	300
TA	C		
TC	U	X	110
TC	C		

Gruppenaufgabe 5 (Lösung während der Übung) Erläutern Sie Probleme, die bei der naiven Implementierung von SSL auftreten können. Lesen Sie hierzu beispielsweise `http://www.ietf.org/mail-archive/web/tls/current/msg07553.html` und skizzieren Sie sowohl den traditionellen Angriff mittels eines Botnets sowie den dort neu vorgestellten Angriff mittels Renegotiation. Wie können derartige Angriffe verhindert werden? Können Sie sich ähnliche Angriffe auch im Bezug auf den Einsatz von Datenbanken vorstellen?

Viele DoS Attacken basieren auf der Tatsache, dass durch den Client eine Operation auf dem Server bewirkt werden kann, deren Bearbeitung den Server mehr kostet als es den Client kostet, die Operation auszulösen.
Im Falle eine DDoS Attacke, d.h. eines verteilten Angriffs ist dies nicht zwingend nötig. Vielmehr muss eine große Zahl von Clients zur Verfügung stehen, die eine kostspielige aber nicht notwendigerweise unproportional teure Operation auf dem Server auslösen. Dies lässt sich beispielsweise durch ein Botnetz[a] realisieren, welches eine Webseite gleichzeitig von tausenden verschiedenen Computern ausruft.
Bei SSL bietet es sich an, eine SSL Verbindung von jedem Rechner des Botnets aufzubauen. Hierdurch werden auf dem Server tausende Handshake-Berechnungen durchgeführt, auf jedem Client jedoch nur eine. Die Last auf dem Server steigt potentiell, bis dieser keine neuen Verbindungen mehr annehmen kann, wodurch eine Webseite effektiv nicht mehr erreichbar ist.
Die vorgeschlagene Renegotiation Attacke verzichtet auf den Multiplikator mittels Botnets indem hier ein Client den Server anweist, die Verschlüsselung neu auszuhandeln. Durch die auf Serverseite höheren Kosten dieser Operation[b] kann ein einzelner Client mittels Renegotiation-Request sehr hohe Kosten auf Serverseite verursachen und so das gleiche Resultat erzielen, wie beim zuvor genannten Angriff.

[a] `http://en.wikipedia.org/wiki/Botnet`
[b] `http://www.gossamer-threads.com/lists/apache/dev/398033?do=post_view_threaded\#398033`