Verteilte Algorithmen

Aufgabenblatt 9

(zum 23. Januar 2013)

Gruppe 08
Björn Stabel 222128
Tim Strehlow 316594
Friedrich Maiwald 350570

Aufgabe 9.1: Transaktionen

a) Wozu sind Transaktionen notwendig? Geben Sie mindestens ein Beispiel.

Transaktionen bezeichnen mehrere Zugriffe, die zusammen genommen entweder fehlerfrei und vollständig, oder im Fehlerfall gar nicht ausgeführt werden, und somit als logische Einheit betrachtet werden können. Damit wird nach deren Ausführung das System in einem konsistenten Zustand hinterlassen.
Ein typisches Beispiel sind Zugriffe auf Datenbanksysteme. Tritt während eines Datenbankzugriffs ein Fehler auf, sodass die Transaktion nicht fehlerfrei zu Ende geführt wird, werden alle bis dahin während der Transaktion vorgenommen einzelnen Änderungen wieder rückgängig gemacht, um den Zustand vor Beginn der Transaktion wieder herzustellen.
Ein anderes abstrakteres Beispiel wäre eine Überweisung, wobei die Transaktion aus dem Abziehen des Betrages von einem Konto und dem Gutschreiben auf einem anderen Konto besteht. Die Transaktion muss sowohl korrekt als auch als komplett durchgeführt werden. Wenn es etwa bei der Gutschrift auf dem Empfängerkonto zu Problemen kommt, muss der Betrag wieder auf das Absenderkonto gezahlt werden, beziehungsweise von diesem erst gar nicht offiziell abgebucht werden.

b) Erklären Sie die ACID Eigenschaften und deren Relevanz.

A – Atomicity: Alles oder Nichts – Die atomaren Operationen einer Transaktion werden entweder komplett ausgeführt oder gar nicht.
C – Consistency: Eine Transaktion überführt das System von einem konsistenten Status wieder in einen konsistenten Status.
I – Isolation: Zwischenstände innerhalb einer Transaktion, die durch die atomaren Operationen erreicht werden, sind außerhalb der Transaktion nicht sichtbar.
D – Durability: Der Status nach erfolgreichem Abschluss einer Transaktion ist auf jeden Fall dauerhaft gespeichert.

c) Wodurch können diese Eigenschaften gefährdet werden? Nennen Sie Beispiele, die zur Verletzung führen.

Gleichzeitig ablaufende Transaktionen können zum Beispiel die genannten Eigenschaften gefährden, wenn diese auf dieselben Daten zugreifen. Laufen die Transaktionen nicht korrekt isoliert von einander, greifen sie auf zwischenzeitlich geänderte Daten zu. Das kann zu falschen Ergebnissen und im Fall eines Rollbacks zu inkonsistenten Datenzuständen führen.
Auch fehlerhafte Umgebungen können Transaktionen gefährden. Wenn die Software- oder Hardware-Umgebung nicht fehlerfrei arbeitet, können beispielsweise Ergebnisse von eigentlich erfolgreichen Transaktionen nicht auf die Festplatte geschrieben und somit permanent gespeichert werden.
In verteilten Datenbanken ist es schwieriger, die ACID-Eigenschaften zu garantieren, da hier bei einer verteilten Transaktion mehrere Instanzen über diese wachen müssen und zur korrekten

Abwicklung der Transaktion sich möglicherweise auf eine vergleichsweise unzuverlässige Netzwerkverbindung verlassen müssen.

Aufgabe 9.2: Eigenschaften von Plänen

a) Was ist der Unterschied zwischen einem serialisierbaren Plan (serializable schedule) und einem seriellen Plan (serial schedule)?

Ein serieller Plan führt die einzelnen Operation pro Transaktion zusammengefasst hintereinander aus (zuerst alle Operationen von Prozess 1, anschließend alle von Prozess 2,...).
In einem serialisierbaren Plan muss das so nicht gegeben sein, allerdings muss zu ihm ein äquivalenter serieller Plan existieren, so dass sein Ergebnis das selbe ist.

b) Zeigen Sie, dass der Einsatz von Two Phase Locking (2PL) nicht immer zu einem striktem Plan (strict schedule) führt!

Ein Plan ist strikt, wenn keine Transaktion Daten liest/überschreibt, die noch nicht commited wurden.
Im 2PL werden einzelnen Transaktionen Locks zugeteilt (Lese- oder Schreib-Locks). Wenn eine Transaktion ein Lock freigibt, kann es keinen neuen Lock beantragen. Lese-Locks können gleichzeitig mehrere vergeben werden (Shared Lock), wenn aber ein Schreib-Lock zugesichert wurde, können keine weiteren Locks mehr vergeben werden (exclusive lock).
Beim einfachen 2PL kann eine Transaktion den Schreib-Lock schon vor dem Ende wieder freigeben, wodurch wieder Lese-Locks an andere Transaktionen vergeben werden können. Dadurch können von verschiedenen Transaktionen Lesezugriffe auf beschriebene Daten, die noch nicht commited wurden ausgeführt werden, was die Eigeschaft eines strikten Planes verletzen würde.

c) Warum resultiert aus dem Einsatz des strikten 2PL immer ein strikter Plan (strict schedule)?

Im strikten 2PL werden Schreib-Locks immer bis zum Ende einer Transaktion (commit/abort) gehalten. Dadurch kann der in b) beschriebene Fall nicht auftreten, wodurch sich ein strikter Plan ergibt.

d) Warum müssen beim strikten 2PL nur Schreib-Sperren (write locks) bis zum Ende der Transaktion gehalten werden?

Reine Lese-Zugriffe ändern den Zustand des Systems nicht, Schreib-Zugriffe schon, deswegen muss der Lock bis zum Commit, erhalten bleiben, da zu diesem Zeitpunkt erst die Änderungen komplett übernommen werden.

Aufgabe 9.3: Lock Escalation

Was ist Lock Escalation und wie kann sichergestellt werden, dass die verschiedenen Locks stets konsistent sind und sich nicht widersprechen?

Mit *Lock Escalation* wird ein Verfahren genannt, um die Granularität von Locks anzupassen. Wenn normalerweise eine feine Granularität für Locks benutzt wird (bei Festplattenzugriffen beispielsweise auf Datei-Ebene), dann kann das bei größeren Transaktionen zu vielen Locks und damit einem hohen Aufwand führen. Als Gegenmaßnahme wird die Lock-Granularität dynamisch erhöht (beispielsweise auf Ordner-Ebene) und damit der verbundene Aufwand wieder gesenkt.
Damit Locks sich generell nicht widersprechen, müssen sie zu einander kompatibel sein. Nach einem read lock für ein Datum dürfen danach nur weiteren read locks erteilt werden, keine write locks (*shared lock*). Wenn ein write lock gewährt wurde, darf kein anderes Lock mehr für das entsprechende Datum erteilt werden (*exclusive lock*). Außerdem können in bestimmten Fällen read locks zu write locks hochgestuft werden beziehungsweise umgekehrt heruntergestuft werden. Diese Regln sorgen dafür, dass die Locks insgesamt konsistent bleiben und Transaktionen auch bei mehreren parallelen Locks immer fehlerfrei durchgeführt werden können.