UNIVERSITÄT SIEGEN

Fakultät IV
Betriebssysteme und verteilte Systeme
Prof. Dr. rer. nat. Roland Wismüller

Aufgabenblatt 10
Musterlösung

Vorlesung Verteilte Systeme
Sommersemester 2020

Aufgabe 1: Vollständig sortierter Multicast

Ein zweiter Ansatz ist, die Nachricht sofort per Multicast zu versenden, aber die Auslieferung zu verschieben, bis der Sequenzer eine Sequenznummer dafür gemulticastet hat. Das passiert, nachdem die Nachricht vom Sequenzer empfangen wurde.

Ein dritter Ansatz ist, zuerst eine Sequenznummer vom Sequenzer zu ermitteln, und die Nachricht dann zu multicasten.

Der erste Ansatz (die Operationen an den Sequenzer zu senden) beinhaltet das Senden einer Punkt-zu-Punkt-Nachricht mit der Operation sowie eine Multicast-Nachricht.

Für den zweiten Ansatz braucht man zwei Multicast-Nachrichten: eine mit der Operation und eine mit der Folgenummer.

Der dritte Ansatz kostet eine Punkt-zu-Punkt-Nachricht mit der Sequenznummer, gefolgt von einer Multicast-Nachricht, die die Operation enthält.

Aufgabe 2: Vollständig sortierter Multicast / Lamport Zeitstempel

Nein, es ist ausreichend, einen beliebigen anderen Nachrichtentyp zu multicasten, so lange diese Nachricht einen Zeitstempel aufweist, der grösser als der der empfangenen Nachricht ist.

Aufgabe 3: Transaktionen

Transaktionen sind ein eng mit den Algorithmen zum wechselseitigen Ausschluss verbundenes Konzept, da sie, ebenfalls wie diese Algorithmen, eine gemeinsam genutzte Ressource gegen den gleichzeitigen Zugriff mehrerer Prozesse schützen. Des weiteren können Transaktionen einem Prozess erlauben, innerhalb einer einzigen atomaren Operation auf mehrere Datenelemente zuzugreifen und diese zu verändern. Wird der Prozess während der Transaktion unterbrochen, wird alles so wiederhergestellt, wie es vor Beginn der Transaktion ausgesehen hat.

a) Die wichtigsten Eigenschaften von Transaktionen sind:

- Atomicity - Für die Aussenwelt erfolgt die Ausführung der Transaktion unteilbar
- Consistency - Die Transaktion verletzt keine System-Invarianten
- Isolation - Nebenläufige Transaktionen beeinflussen sich nicht gegenseitig.
- Durability - Nachdem eine Transaktion festgeschrieben ist, bleiben die Änderungen permanent.

b) Ein Deadlock kann im Zusammenhang der Nebenläufigkeit auftreten, wenn zwei Transaktionen gegenseitig auf die Freigabe einer Ressource warten. Da das Ziel der Nebenläufigkeit die Ausführung mehrerer Transaktionen zur gleichen Zeit und dabei die Beibehaltung eines konsistenten Status aller Datenelemente ist, ist hier eine Nebenläufigkeitskontrolle notwendig.

Aufgabe 4: Zwei-Phasen-Commit

Ein Teilnehmer könnte in seinem INIT-Status auf eine VOTE-REQUEST-Nachricht vom Koordinator warten. Wird diese Nachricht nicht nach einer bestimmten Zeit empfangen, beschliesst der Teilnehmer einfach, die Transaktion lokal abzubrechen und sendet eine VOTE-ABORT-Nachricht an den Koordinator.
Analog kann der Koordinator im Status WAIT blockieren, wo er auf die Abstimmung der anderen Teilnehmer wartet. Können innerhalb einer bestimmten Zeit nicht alle Abstimmungen ermittelt werden, sollte der Koordinator ebenfalls entscheiden, dass die Transaktion abgebrochen wird und ein GLOBAL-ABORT an alle Teilnehmer senden.
Der dritte Status, in dem blockiert werden kann, ist der READY-Status seitens der Teilnehmer. Hier könnte blockiert werden, da ein Teilnehmer auf das globale Abstimmergebnis wartet, welches vom Koordinator versandt wird. Wird diese Nachricht nicht innerhalb einer bestimmten Zeit empfangen, kann der Teilnehmer nicht einfach entscheiden, die Transaktion abzubrechen. Stattdessen muss er feststellen, welche Nachricht der Koordinator gesendet hat. Am einfachsten löst man dieses Problem, in dem jeder Teilnehmer blockiert, bis der Koordinator wiederhergestellt ist.

Aufgabe 5: Konsistenzmodelle

a) Da für viele Anwendungen eine strikte Konsistenz nicht immer notwendig ist, wurden schwächere Konsistenzmodelle eingeführt. Je schwächer das Konsistenzmodell wird, desto einfacher und effizienter wird die Implementierung der Speicherverwaltung, aber der Aufwand für den Programmierer steigt.

Schwache Konsistenzmodelle entstammen der Notwendigkeit, zur Leistungsteigerung Repliken zu verwenden. Eine effiziente Replikation ist jedoch nur möglich, wenn wir globale Synchronisierungen vermeiden können, was wiederum erreicht werden kann, indem die Konsistenzbeschränkungen gelockert werden.

b) linker Ablauf:

- Der Ablauf ist **nicht** strikt konsistent, weil P_2 noch einen alten Wert von x liest.
- Sequentiell konsistent heißt, dass alle Prozesse alle Zugriffe in derselben Reihenfolge sehen (nicht notwendigerweise der Zeit nach sortiert), bzw., daß der Ablauf durch eine beliebige Verzahnung der Zugriffe zustandekommen kann, in der die Zugriffe für jeden Prozeß in Programmordnung auftreten. Der Ablauf ist **nicht** sequentiell konsistent, da es keine solche Verzahnung geben kann:
  - $R_3(x)b$ muß in jedem Fall nach $W_1(x)b$ auftreten (sonst würde P_3 ein a lesen),
  - $R_2(y)a$ muß aus dem selben Grund nach $W_3(y)a$ auftreten,
  - $W_1(x)a, W_1(x)b$ sowie $R_2(y)a, R_2(x)a$ und $R_3(x)b, W_3(y)a$ müssen jeweils in dieser Reihenfolge stehen, da sie im selben Prozeß stattfinden.

  Aus Gründen der Transitivität folgt damit aber, daß $R_2(y)a$ in der Verzahnung **nach** $W_1(x)b$ stehen muß, was ein Widerspruch ist (es müßte dann b gelesen werden).
- Kausal konsistent heißt dass alle Prozesse kausal abhängige Zugriffe in derselben Reihenfolge sehen. Der Verlauf ist **nicht** kausal konsistent:
  - $W_3(y)a$ ist kausal abhängig von $W_1(x)b$ (wegen der Kommunikation zwischen P_1 und P_3 über die Variable x: $W_1(x)b$ und $R_3(x)b$),
  - P_2 sieht diese Schreibzugriffe aber in umgekehrter Reihenfolge: er sieht bereits $y = a$, bevor er $x = b$ sieht (der Lesezugriff $R_2(x)a$, den P_2 nach $R_2(y)a$ ausführt, liefert immer noch den alten Wert a.

rechter Ablauf:

- Der Verlauf ist nicht strikt konsistent, weil P_3 noch einen alten Wert von x liest ($R_3(x)a$).
- Der Verlauf ist sequentiell konsistent: eine mögliche Verzahnung der Zugriffe ist z.B. $W_1(x)a, R_2(x)a, R_3(x)a, W_3(y)a, W_2(x)c, R_2(y)a, R_3(x)c, W_1(x)b, R_3(x)b$:

P1:	W(x)a								W(x)b
P2:		R(x)a				W(x)c	R(y)a		
P3:			R(x)a	W(y)a				R(x)c	R(x)b

- Der Ablauf ist kausal konsistent. Kausale Abhängigkeiten treten auf zwischen $W_1(x)a$ und $W_2(x)c$ sowie zwischen $W_1(x)a$ und $W_3(y)a$. In beiden Fällen sehen alle Prozesse dieselbe Reihenfolge. $W_1(x)b$ und $W_2(x)c$ sind kausal **un**abhängig, so daß es kein Problem ist, daß P_3 zuerst $x = c$ und dann $x = b$ sieht.
  (Der Ablauf muß auch schon deshalb kausal konsistent sein, weil er sequentiell konsistent ist.)

Aufgabe 6: Sequentielle Konsistenz

P_1 liest den Wert von $x = 2$, bevor er den Wert von y auf 1 setzt. P_2 aber setzt x auf 2 erst nachdem er $y = 1$ gelesen hat (y war vorher Null). Deswegen sind diese zwei Ausführungen inkompatibel und das Speicherverhalten ist nicht sequentiell konsistent.

Aufgabe 7: Kausale Konsistenz

a) P_2: Durch das Lesen von x und das Schreiben von y entsteht eine kausale Abhängigkeit. D.h. $W_1(x)1$ ist kausal vor $W_2(y)2$. Um das Verhalten bei P_3 zu erklären, müßte die globale Reihenfolge aber $W_2(y)2$, $R_3(y)2$, $R_3(x)0$, $W_1(x)1$ sein, was dazu im Widerspruch steht ($W_1(x)1$ nach $W_2(y)2$).

b) Die kausale Konsistenz ist möglicherweise ausreichend. Das Problem ist, dass die Reaktionen auf Änderungen der Aktienwerte konsistent sein sollten. Änderungen von unabhängigen Aktienwerten können in unterschiedlichen Reihenfolgen angezeigt werden.