Inhaltsverzeichnis

1 Einleitung

- Transaktionen sind kritischer Bestandteil von vielen Anwendungen - In verteilten Systemen ist Konsistenz schwierig zu erreichen - Microservices bedeutet: Aufteilung des Systems in verschiedene Domänen - Fehlerbehandlung wird komplex - Ein System sollte sein: responsiveness, resilience, elasticity and asynchronous message passing - außerdem: ACID Garantie - 2PC: blockierend
alles aus Quelle The Saga Pattern in a Reactive Microservices Environment

2 System, Konsistenz, Lösungsstrategien

2.1 Grundlagen

Um das in dieser Arbeit betrachtete Saga-Pattern zu verstehen, sollen zuerst einige Grundlagen erläutert werden. Besonders die im Titel der Arbeit enthaltenen Begriffe *System* und *Konsistenz* sollen in diesem Abschnitt erläutert werden.

2.1.1 System allgemein

Ein System beschreibt einen abgegrenzten Bereich der objektiven Realität. Außerhalb dieses Bereichs liegt die Umgebung, die somit nicht zum System gehört. Zwischen des Systems und seiner Umgebung befindet sich der Systemrand.

2.1.2 System in der Softwareentwicklung

In der Softwareentwicklung besteht ein System aus einer Menge miteinander interagierenden Softwarekomponenten. Diese Komponenten arbeiten an einem gemeinsamen Ziel. Neben der Software und deren Quellcode gehören auch Nutzerhandbücher, Tests, Bestandteile für die Instandhaltung sowie Spezifikationen und Konzepte zum System. A

2.1.3 Zustand von Systemen

Ein Softwaresystem befindet sich zu jedem Zeitpunkt in einem Zustand. Der Wechsel eines Zustands ist die Folge von Nutzerinteraktionen und festgelegten Routinen. Damit das System reibungslos funktionieren kann, darf es nur zwischen gültigen Zuständen wechseln.

2.1.4 Sichten auf Systeme

Softwaresysteme können unter verschiedenen Sichten betrachtet werden. TODO
- mathematische Sicht: Endlicher Automat - Endliche Menge von Zuständen - Endliches Alphabet - Übergangsfunktion von Zustand x Alphabet -> Zustand - Startzustand - endliche Menge von Endzuständen

2.1.5 Transaktionen

Ein Übergang von einem Zustand in den nächsten Zustand im Modell des Zustandsautomats ist eine einzige Operation. Häufig äußert sich eine solche Operation jedoch in einer Folge von auszuführenden Operationen. Eine solche Folge von Operationen wird

als Transaktion bezeichnet. Teil einer Transaktion kann beispielsweise eine oder mehrere Datenbankoperationen oder der Aufruf einer externen Schnittstelle sein. Transaktionen können auf verschiedenen Ebenen betrachtet werden. Auf Datenbankebene werden Transaktionen anders umgesetzt als auf Ebene der Geschäftslogik.

2.1.6 ACID Eigenschaft von Transaktionen

ACID ist ein Akronym, welches häufig verwendet wird, um Eigenschaften auszudrücken, die eine Transaktion auf Datenbankebene aufweisen sollte. Die Buchstaben stehen für *A*tomicity (Atomarität), *C*onsistency (Konsistenz), *I*solation (Isolation) und *D*urability (Dauerhaftigkeit).
Atomarität beschreibt hier das Alles-Oder-Nichts-Prinzip. Eine Transaktion, also eine Folge von Datenbankoperationen, muss entweder alle Operationen ausführen oder in einem Fehlerfall alle Änderungen auf den ursprünglichen Zustand zurückrollen. Als illustratives Beispiel soll das Szenario einer Überweisung dienen. Eine Überweisung besteht aus dem Erhöhen des Kontostands des Debitors und dem Verringern des Kontostands des Kreditors. Eine solche Transaktion kann fehlschlagen. Dabei ist sicherzustellen, dass die initialen Kontostände wiederhergestellt werden. Wird die Atomarität nicht gewährleistet, könnte sich die Gesamtsumme des im System enthaltenen Systems verändern, Geld geht also verloren oder Geld wird aus dem Nichts generiert. Das ist natürlich nicht akzeptabel.
Konsistenz lässt sich mit widerspruchsfrei übersetzen. War das System vor Ausführung der Transaktion widerspruchsfrei, also konsistent, dann ist es das nach Ausführung der Transaktion ebenso. Um dies zu gewährleisten, sind verschiedene Mechanismen im Datenbanksystem im Einsatz. Dies werden im folgenden Abschnitt beschrieben.
Isolation stellt sicher, dass sich parallel in Ausführung befindliche Transaktionen nicht gegenseitig beeinflussen können. Das bedeutet, dass eine Transaktion Teile der Datenbasis sperren können muss. Eine solche Sperre führt jedoch zu blockierendem Verhalten und schränkt somit die nebenläufige Benutzung der Datenbank ein.
Dauerhaftigkeit von Transaktionen muss gewährleistet sein, damit bei einem Ausfall Transaktionsdaten nicht verloren gehen. Um Dauerhaftigkeit zu erreichen, wird ein Transaktionslog gefährt, welches im Ausfall verwendet werden kann, um die verlorenen Transaktionen nachzuspielen.

2.1.7 Konsistenz auf Datenbankebene

Auf Datenbankebene wird die Konsistenz durch drei Mechanismen sichergestellt: Bereichsintegrität, Entitätsintegrität, Referentielle Integrität und logische Konsistenz.
Die Bereichsintegrität stellt lediglich sicher, dass die Werte einer Spalte in einem definierten Wertebereich befinden. Die Entitätsintegrität beschreibt die Eindeutigkeit eines Datensatzes und somit Fähigkeit, diesen Datensatz mittels Primärschlüssel eindeutig identifizieren zu können.
Die Referentielle Integrität gewährleistet, dass Relationen zwischen Datensätzen nicht auf nicht existierende Datensätze zeigen können. Ist ein Fremdschlüssel gesetzt, so muss der referenzierte Datensatz existieren.
Die in der Datenbank enthaltenen Daten drücken einen Sachverhalt aus. Die semantische Bedeutung dieser Daten kann nicht durch das Datenbanksystem sichergestellt werden. Es

liegt in der Verantwortung des Entwicklers, allen Nutzerinteraktionen mit dem System die korrekten Transaktionen zuzuordnen. Ein System ist logisch konsistent, wenn kein logisch inkonsistenter Zustand erreichbar ist.

2.1.8 Konsistenz auf Systemebene

2.1.9 Begriffsdefinition Konsistenz im Zusammenhang mit dem System

- System muss als DFA modelliert sein - Durchführung einer Aktion ist eine Übergangsrelation - Übergangsrelation kann folge von atomaren Transaktionen sein

2.2 Lösungsstrategien

2.2.1 Instrumente für Sicherstellung von Konsistenz in Monolithischen Anwendungen

- Definieren von Transaktionen - Folge von Anweisungen, die zusammen ausgeführt werden müssen - Commit oder Rollback - Unterstützung von Lokalen Transaktionen durch Datenbank-Transaktionen

2.2.2 Instrumente für Sicherstellung von Konsistenz in Verteilten Systemen

- Transaktion beinhaltet Aktion, die eine Abhängigkeit aufruft (zB Aufruf einer Http-Schnittstelle) - Zentrales Problem: Wie stelle ich sicher, dass ein Aufruf geklappt hat? Wie gehe ich vor, wenn eine Aktion einer Transaktion nicht geklappt hat? - 2 Phasen Commit als verteilte Umsetzung des Transaktionsvorgehens - Beschreibung - Nachteile: sehr hohe Chattines, sehr langsam, blockierend, geringer Throughput, komplexe Implementierung

2.2.3 Saga-Pattern

- Fehlerbehandlungsstrategie für monolithische und verteilte Systeme - Auflösen der atomarität der Transaktionen in einzelne lokale Transaktionen T - Definieren von Kompensationsaktionen C - jedes T hat ein C - sequentielle Ausführung der Ts - schlägt ein T fehl, kann ein entsprechendes C ausgeführt werden

3 Saga Pattern

3.1 Grundprinzipien

- Saga ist erfolgreich, wenn alle Ts erfolgreich ausgeführt werden -> Übergang von: Ausgangszustand -> Ausführung T1 -> Ausführung T2 -> Endzustand - Saga ist fehlgeschlagen, wenn ein T fehlschlägt - Ausführung der Cs - Systemzustand ist danach immer noch konsistent - Backward Recovery: Alle Ts, die ausgeführt wurden, werden durch Ausführung des entsprechenden Cs gerollbackt - Forward Recovery: Einführung von Save Points zwischen den Ausführungen (zB T1-T5, Checkpoint nach T2 und Checkpoint nach T3) - Fehler in T3 führt zu Rollback bis letztem Save Point S1 (entspricht Ausführung von C3, Zustand nach Ausführung nach T1 und T2) - Wiederaufnehmen der Saga: Ausführung von T3 - T5 - Kompletter Rollback, falls es nicht geht (Backward Recovery)

3.1.1 Ts und Cs

3.1.2 Forward Recovery

3.1.3 Backward Recovery

3.1.4 Saga Execution Component

3.1.5 Transaktionslog

3.2 Orchestration und Choreographie

3.3 Asynchronität

4 Design

4.1 Design Choreografie

4.2 Design Orchestration

5 Implementierung

Abbildungsverzeichnis