Exposé

Entwicklung eines verteilten Nachrichtensystems mit optimistischer Replikation

Anton Altenbernd (351344)

1. Motivation

Die Replikation von Daten ist eine wichtige Technologie für verteilte Dienste. Dabei wird der Dienst und die zugehörigen Daten auf mehreren Computern repliziert, die im folgenden Replika genannt werden. Dies verbessert die Verfügbarkeit, die Performance und den Durchsatz eines Dienstes.

Über populäre Kommunikationsdienste, wie E-Mail, WhatsApp, Twitter usw., werden täglich mehrere Milliarden Nachrichten verschickt. Die Bearbeitung von Daten in diesem Ausmaß können von einem Dienst nur bearbeitet werden, wenn dieser repliziert wird, d.h. die Anfragen der Nutzer werden von mehreren Servern bearbeitet und die Daten werden konsistent auf allen Servern gehalten.

In der pessimistischen, klassischen Replikation von Daten wird der Zugriff auf Server blockiert bis diese den aktuellen Zustand haben. Beispielsweise wird eine primär-Replika bestimmt, die dafür verantwortlich ist, dass alle anderen Replikas aktuell bleiben. Nach einem Update übergibt die primär-Replika die Änderungen an die sekundär-Replikas. Falls die primär-Replika abstürzt, wählen die übrigen Replikas eine neue primär-Replika. Diese Vorgehensweise funktioniert sehr gut in lokalen Netzwerken, in denen Verzögerungen klein und Fehler sehr selten sind. In einer Umgebung wie dem Internet, in dem Verzögerungen sehr groß sind, ist mit großen Performance- und Verfügbarkeitsverlusten zu rechnen.

In der optimistischen Replikation von Daten hingegen wird davon ausgegangen, dass Probleme sehr selten, wenn überhaupt, auftreten. Updates werden im Hintergrund ausgeführt und Konflikte werden erst, nachdem sie aufgetreten sind, aufgelöst. Dies führt dazu, dass das System nicht blockieren muss und effizient auch bei schwacher oder instabiler Netzwerkverbindung arbeiten kann.

2. CRDT

Konsistenz ist ein wichtiger Bestandteil in der Umsetzung von verteilten Systemen. Nach dem CAP-Theorem kann ein verteiltes System jedoch nur zwei der folgenden Eigenschaften erfüllen: Konsistenz, Verfügbarkeit und Partitionstoleranz. Die Konsistenz steht also im Konflikt zur Verfügbarkeit und der Partitionstoleranz.

„Eventual Consistency“ ist ein Konsistenzmodell, in dem ein Update auf einer Replika ausgeführt wird und erst später auf die anderen Replikas verteilt wird. Alle Updates werden irgendwann auf den anderen Replikas ausgeführt. Nebenläufige Updates können jedoch Konflikte hervorrufen, sodass die Updates in falscher Reihenfolge ausgeführt werden. Um diese Konflikte zu lösen, werden komplexe Algorithmen benötigt.

Die Kernidee von CRDTs ist, dass alle Operationen kommutativ und idempotent sind. Durch diese Eigenschaft können alle Operationen in beliebiger Reihenfolge ausgeführt werden. Da Replika irgendwann alle Operationen erhalten, befinden sich dadurch alle Replikas im selben Zustand.

3. Anwendung

Im Rahmen dieser Bachelorarbeit soll ein Prototyp eines verteilten Nachrichtendienstes mit optimistischer Replikation implementiert werden. Die Umsetzung erfolgt in Form eines E-Mail-Dienstes. Die optimistische Replikation wird hier mittels der oben beschriebenen Datenstruktur CRDT realisiert.

Die Anwendung wird als stateful-Webanwendung mit persistenter Datenspeicherung entwickelt, sodass die Replika sich nach einem Absturz noch im selben Zustand befindet. Als Technologie wird das Python-Framework Django verwendet, da dieses bereits die nötigen Funktionen mitliefert und so auf einer höheren Ebene ansetzt.

Im folgenden werden die Funktionalitäten des verteilten E-Mail-Dienstes aufgelistet:

- Anlegen von Benutzerkonten
- Benutzeroberfläche
- Hinzufügen und Löschen von Nachrichten
- Senden und Empfangen von Nachrichten
- Halten von Nachrichten in einer Ordnerstruktur
- Verschieben von Nachrichten in der Ordnerstruktur
- Anlegen und Löschen von Ordnern

4. Experiment

Im empirischen Teil der Bachelorarbeit sollen die Stärken und Schwächen der oben beschriebenen Datenstruktur CRDT ausgearbeitet und anhand des E-Mail-Dienstes getestet werden. Dabei sollen CRDTs hinsichtlich Skalierbarkeit und Konvergenzverhalten geprüft werden. Außerdem soll der Overhead an Daten, der von CRDTs benötigt wird, mittels „batching“ optimiert werden. D.h. Updates werden nicht einzeln sondern in größeren Paketen gesammelt und anschließend versendet.

5. Zeitplan

Woche	
1. bis 2.	Recherche CRDTs, Recherche Skalierung, Implementierung I: Hinzufügen/Löschen
3. bis 4.	Implementierung II : Hinzufügen/Lösen, Benutzerkonten, GUI
5. bis 6.	Implementierung III : Ordnerstruktur
7. bis 8.	Entwicklung der Testumgebung
9. bis 10.	Ausarbeitung der theoretischen Hintergründe
11. bis 12.	Dokumentation der Webanwendung und dazugehörigen Testumgebung

6. Literatur

(1) Saito, Yasushi, and Marc Shapiro. "Optimistic replication." *ACM Computing Surveys (CSUR)* 37.1 (2005): 42-81.

(2) Shapiro, Marc, et al. "Conflict-free replicated data types." *Stabilization, Safety, and Security of Distributed Systems*. Springer Berlin Heidelberg, 2011. 386-400.

(3) Letia, Mihai, Nuno Preguiça, and Marc Shapiro. "CRDTs: Consistency without concurrency control." *arXiv preprint arXiv:0907.0929* (2009).