Anmerkungen zum Paper

Horst Schneider

21. Mai 2014

Inhaltsverzeichnis

1 Abstract **2**

2 Introduction **2**

3 Replicated Data Types **4**

1 Abstract

- Geo-replicated Datenbanken garantieren sofortige Verfügbarkeit und Partitionstoleranz auf Kosten schwacher Konsistenz -> eventual consistency
- es gibt einige Verwirrung über die Semantik von eventual consistency
- die Vergleichbarkeit verschiedener Semantiken von eventual consistency ist unmöglich, da die Semantik oft nur implizit durch die Implementierung gegeben ist bzw. keine Formalismen bei der Beschreibung verwendet werden
- eventual consistency ist verbunden mit verschiedenen Features wie conflict resolution policies, session guarantees, casuality guarantees und Konsistenzleveln

Ziel des papers ist es, ein Framework zur deklarativen Spezifikation von eventual consistency zu bieten, das auf Axiomen basiert.

2 Introduction

- Moderne Internetanwendungen basieren auf verteilten Datenbanksystemen, die oft geo-replicated sind
  - geo-replicated Datenbanken erfordern **partition tolerance**
  - die Anwender fordern **sofortige Verfügbarkeit**
  - -> nach dem CAP-Theorem kann daher keine starke Konsistenz gewährleistet werden
- eventual consistency beschreibt die **Garantie**, dass die Datenbank irgendwann in einen konsistenten Zustand kommt, wenn keine Updates mehr die Datenbank erreichen
- Im Gegensatz zu relationalen Datenbanken sind die Konsistenzmodelle von geo-replicated Datenbanken nicht eindeutig und ausreichend studiert
- Der Ausruck eventual consistency ist nicht eindeutig besetzt und wird als catch-all buzzword verwendet

- Spezifizierungen von geo-replicated Konsistenzmodellen sind aus verschiedenen Gründen unzureichend:
  - verschiedene, unvereinbare Formalismen bzw. keine Formalismen, oft an spezifische Implementierungen gebunden
  - schwache Garantien; eventual consistency bezeichnet oft nur quiescent consistency, einem theoretischen Ansatz, der nicht den praktischen Fall berücksichtigt, dass eventuell nie der Zustand eintritt, in dem keine neuen Updates an der Datenbank ankommen
  - Schwierigkeit von Conflict Resolution Policies: Vorgehen, wenn zwei Replicas beim Synchronisieren unterschiedliche Versionen des gleichen Objektes besitzen
  - Kombination verschiedener Konsistenzlevel: manche Systeme bieten die Mischung von eventual consistency und strong consistency (z.B. Amazon dynamoDB). Ebenso gibt es Implementierungen für Transaktionen in geo-replicated Systemen. Die Semantik hinter verschiedenen Konsistenzleveln und deren Zusammenspiel ist schwierig zu erfassen.
- ein einheitlicher Formalismus soll die Beantwortung folgender Fragestellungen erleichtern:
  - Erfüllen die Anforderungen an meine Anwendung die gewünschte Ausprägung von eventual consistency?
  - Kann ich einen replizierten Datentyp aus System X in System Y verwenden?
  - Welche Semantik steckt hinter der Kombination verschiedener Formen von eventual consistency?
- Die Spezifikation eines eventual consistency models besteht aus folgenden Komponenten:
  - Spezifikation verschiedener Datentypen (Register, Zähler, multi-value Register (in [17]: binary blobs mit key) observed-remove sets) und deren conflict resolution policy auf Objektebene
  - Spezifikation der Datenbankweiten Konsistenzgarantien anhand von Axiomen
  - Interface-Spezifikation: gibt den Anwendern die Möglichkeit, Konsistenzlevel für Operationen, gebündelte Operationen (fences) und Transaktionen festzulegen (bspw. mit Annontationen)
- verschiedene exisitierende Formen von eventual consistency werden systematisch in ein Spezifikationsframework überführt, das Garantien und Features der verschiedenen Technologien einheitlich ausdrücken kann und so Vergleichbarkeit schafft bzw. Interaktionen zwischen Systemen beschreiben kann
- die Spezifikation wird durch eine abstrakte Implementierung, basierend auf Algorithmen in existierenden Systemen, validiert

- die Spezifikation soll stärker als quiescent consistency sein und auch in dem Fall zutreffen, wenn kontinuierlich Updates an der Datenbank ankommen
- Das Paper beinhaltet Beweise, dass die Spezifikation die rigorose Vergleichbarkeit zwischen Features und Garantien gewährleistet

3 Replicated Data Types

Grundideen:

- Replizierte Datenbank speichert Objekte aus der Menge $Obj = \{x, y, ...\}$
- Jedes Objekt besitzt einen Wert aus dem Set Val
- Jedes Objekt $x \in Obj$ besitzt einen Typ type$(x) \in Type$, der die Menge $Op_{type(x)}$ von Operationen festlegt, die Clients auf dem Objekt ausführen können
- Beispiele:
  - Datentyp Counter mit den Operationen *read* und *increment*: $Op_{ctr} = \{rd, inc\}$
  - Datentyp Integer-Register mit den Operationen *read* und *write*: $Op_{ctr} = \{\text{rd}\} \cup \{\text{wr}(k) | k \in \mathbb{Z}$
- Die Semantik von sequenziell ausgeführten Operationen eines Datentyps τ in einem strongly consistent System kann mit folgender Funktion ausgedrückt werden: $S_\tau : Op_\tau^+ \rightarrow Val$
- σ beschreibt eine Sequenz beliebiger Operationen
- $\perp$ beschreibt Operationen, die keinen Wert zurückgeben
- Beispiele:

$$S_{ctr}(\sigma \text{ rd}) = (\text{the number of inc operations in } \sigma); \quad (1)$$

$$S_{intreg}(\sigma \text{ rd})) = k; \text{if wr}(0)\ \sigma = \sigma_1 \text{ wr}(k)\ \sigma_2 \text{ and } \sigma_2 \text{ does not contain wr operations}; \quad (2)$$

$$S_{ctr}(\sigma \text{ wr}(k)) = S_{intreg}(\sigma \text{ inc}) = \perp \quad (3)$$

Glossar

eventual consistency bezeichnet die Einschränkung, dass Daten nach dem CAP Theorem zwar sofort verfügbar und partition tolerant sind, dafür aber nicht sofort konsistent sind, sondern erst an einem unbestimmten Punkt in der Zukunft. 2, 3

geo-replicated bezeichnet die räumlich voneinander getrennte Verteilung von replizierten (nicht-relationalen) Datenbanken. 2

quiescent consistency basiert auf den zwei Prinzipen "Method calls should appear to happen in a one-at-a-time, sequential order" und "Method calls separated by a period of quiescence should appear to take effect in their real-time order". Ein Method Call ist definiert als die Sequenz *invocation*, *pending* und *response*. Prinzip 1 besagt, dass gleichzeitige invocations, die die gleichen Werte manipulieren, selbst bei überlappender pending Phasen keine Mischung ihrer Ergebnisse erzeugen sollten. Man weiß zwar nicht, welcher Call zuerst abgeschlossen ist, aber die Ergebnisse sollten trotzdem aussehen, als ob eine sequentielle Reihenfolge der Befehle gegeben wäre. Prinzip zwei besagt, dass, wenn es eine *ruhende* Phase (= keine pending Calls) zwischen zwei Method Calls gab, die Ergebnisse der beiden Method Calls in ihrer real-time order vorhanden sein sollten. Bsp.: „For example, suppose A and B concurrently enqueue x and y in a FIFO queue. The queue becomes quiescent, and then C enqueues z. We may not be able to predict the relative order of x and y in the queue, but we know they are ahead of z.“. 3