

# Verteilte Systeme

Prof. Dr. Martin Becke

CaDS - HAW Hamburg

Version 0.9

# Inhalt

- 1 Konsensbildung und Fehlertoleranz
  - Konsensbildung
  - Fehlertoleranz

# Konsensbildung

## Herausforderungen

- ▶ Fehlertoleranz
- ▶ Kommunikationslatenz
- ▶ Sicherheit und Integrität

# Konsensbildung

## Einsatz

- ▶ Datenbanken und verteilte Speichersysteme
- ▶ Blockchain-Technologie
- ▶ Verteilte Rechensysteme

# Konsensbildung

Eine Basis : Quorumsabstimmung

- ▶ Methode zur Erzielung von Konsens
- ▶ Braucht minimale Anzahl von Knoten, die an einer Abstimmung teilnehmen
- ▶ Zustimmung von mehr als  $N/2$
- ▶ Kompromiss zwischen Leistung und Zuverlässigkeit

# Konsensbildung

Grundidee : Quorumsabstimmung

- ▶ Der Koordinator sendet eine Anfrage an alle anderen Knoten, um eine Abstimmung zu initiieren. Diese Anfrage enthält die Informationen, über die abgestimmt werden soll.
- ▶ Jeder Knoten verarbeitet die Anfrage und sendet seine Stimme zurück an den Koordinator.
- ▶ Der Koordinator sammelt alle Stimmen. Wenn die Mehrheit der Stimmen eine Zustimmung ist (d.h., das Quorum erreicht ist), wird die Operation ausgeführt und das Ergebnis an alle Knoten kommuniziert.

# Konsensbildung

## Zentral vs dezentral

- ▶ Zentral :
  - ▶ Braucht häufig « Koordinator » oder « Leader »
- ▶ Dezentral
  - ▶ Komplexer zu implementieren, aber sehr robust

# Konsensbildung

Zentral : ZAB

- ▶ ZooKeeper Atomic Broadcast (ZAB) ist ein Konsensprotokoll
- ▶ ZAB ist ein Crash-Recovery-Protokoll
- ▶ Entdeckungsphase wählen die Knoten einen Anführer
- ▶ Broadcastphase werden Follower informiert
- ▶ Verteiltes Konfigurationsmanagement (recht träge für viele Daten)



# Konsensbildung

De-zentral : Blockchain

- ▶ Blockchain ist wie ein digitales Kassenbuch
- ▶ Transaktionen werden Teilnehmern zur Prüfung vorgelegt
- ▶ Falls gültige Transaktion, wird sie als « Block » gespeichert
- ▶ Integrität der Kette muss geschützt werden  
(Konsensmechanismus)

# Blockchain

## Konsensmechanismus

- ▶ Proof-of-Work (PoW) (komplexe kryptografische Rätsel)
- ▶ Proof-of-Stake (PoS) (ökonomische Anreize)
- ▶ Proof-of-Authority (PoA) :
- ▶ Practical Byzantine Fault Tolerance (PBFT)

# Blockchain

## Probleme 2023

- ▶ Skalierbarkeit - Ansatz Shim Layer
- ▶ Energieverbrauch - (Abkehr von PoW)
- ▶ 51%-Angriffe
- ▶ Datenunveränderlichkeit
- ▶ Komplexität und Benutzerfreundlichkeit

# Fehlertoleranz

## Idee

- ▶ Fehlersituation nicht als Ausnahme ansehen
- ▶ Fehlersituation ein zu erwartendes Ereignis
- ▶ Incident Management

# Fehlertoleranz

## Techniken

- ▶ Redundanz
- ▶ Replikation
- ▶ Wiederherstellung
- ▶ Byzantinische Fehlertoleranz (BFT)

# Fehlertoleranz

## Fehlerarten

- ▶ Hardware-Fehler
- ▶ Software-Fehler
- ▶ Kommunikationsfehler
- ▶ Byzantinische Fehler

# Fehlertoleranz

## Fehlerbegriff

- ▶ Fault (Fehler)
- ▶ Error (Fehlzustand)
- ▶ Failure (Ausfall)

# Fehlertoleranz

## Fehlermodelle

- ▶ Verfügbarkeit
- ▶ Zuverlässigkeit
- ▶ Sicherheit
- ▶ Wartbarkeit



# Fehlertoleranz

## Fehlermodelle

- ▶ Crash failure
- ▶ Omission failure (Auslassungsfehler)
- ▶ Receive omission
- ▶ Send omission
- ▶ Timing failure
- ▶ Response failure
- ▶ Value failure
- ▶ State-transition failure
- ▶ Arbitrary failure

# Fehlertoleranz

Beispiel : Eigenschaften von Halting-Fehler oder Crash-Fehler

- ▶ Dauer des Ausfalls
- ▶ Ursache des Ausfalls
- ▶ Reichweite des Ausfalls
- ▶ Vorhersehbarkeit des Ausfalls

# Fehlertoleranz

## Synchron

- ▶ Strenge Timing-Anforderungen
- ▶ Kann Timeout implementieren
- ▶ Zusätzliche Probleme mit Synchronisation

# Fehlertoleranz

## Asynchron

- ▶ Keine Timing-Anforderungen
- ▶ Techniken wie Sequenznummern oder Quittungen notwendig

# Fehlertoleranz

## Verhaltensweisen und Auswirkungen bei Fehler

- ▶ Fail-Stop-Fehler
- ▶ Fail-Noisy-Fehler
- ▶ Fail-Silent-Fehler (Halteproblem ?)
- ▶ Fail-Safe-Fehler
- ▶ Fail-Arbitrary-Fehler (Übel !)

# Fehlertoleranz

## Redundanz

- ▶ Datenredundanz
- ▶ Rechenredundanz
- ▶ Kommunikationsredundanz

# Fehlertoleranz

## Resilienz

- ▶ Replikation
- ▶ Überwachung
- ▶ Wiederherstellung

# Fehlertoleranz

## Replikation

- ▶ Daten und Dienste duplizieren
- ▶ Erhöht Verfügbarkeit (Rechenredundanz)
- ▶ Erhöht die Leistung (Load-sharing)



# Fehlertoleranz

## Replikationsstrategie

- ▶ Permanente Replikation
- ▶ Serverinitiierte Replikation
- ▶ Clientinitiierte Replikation

# Fehlertoleranz

## Replikationsansätze

- ▶ Passive Replikation (auch Primär-Backup-Replikation oder Master-Slave-Replikation)
- ▶ Replikation Replikation (auch Multi-Master- oder State-Machine-Replikation)

# Fehlertoleranz

Grad der Verfügbarkeit

- ▶ Cold
- ▶ Warm
- ▶ Hot

# Fehlertoleranz

## Absicherung

- ▶  $2k$  Replikation (Crash-Fehler/ fail-stop-Fehler)
- ▶  $2k + 1$  Replikation (Crash-Fehler und Omission-Fehler)
- ▶  $3k + 1$  Replikation (byzantinische Fehler)

# Fehlertoleranz

## Erasure Coding

- ▶ Technik zur Datenredundanz und Fehlerkorrektur
- ▶ Arbeitet mit Paritätsinformationen und Datenaufteilung
- ▶ Keine 1 : 1 Daten-Kopien/Replikationen
- ▶ Erhöhte Komplexität bei Operationen auf Daten
- ▶ Kann  $n - k$  Verluste tolerieren

# Fehlertoleranz

## Erasure Coding - Beispiel $k = 4$

- ▶ Fragment 1 : « AB »
- ▶ Fragment 2 : « CD »
- ▶ Fragment 3 : « EF »
- ▶ Fragment 4 : « GH »

# Fehlertoleranz

## Erasure Coding - Beispiel $n = 6$

- ▶ Zwei zusätzliche Paritätsfragmente ( $n-k=2$ )
- ▶ Hier XOR-Verfahren für Paritätsfragmente
- ▶ Paritätsfragment 1 : Fragment 1 XOR Fragment 2  
 $= \langle AB \rangle \text{ XOR } \langle CD \rangle = \langle P1 \rangle$
- ▶ Paritätsfragment 2 : Fragment 3 XOR Fragment 4  
 $= \langle EF \rangle \text{ XOR } \langle GH \rangle = \langle P2 \rangle$

# Fehlertoleranz

## Erasure Coding - Beispiel 6 Fragmente

- ▶ Fragment 1 : « AB »
- ▶ Fragment 2 : « CD »
- ▶ Fragment 3 : « EF »
- ▶ Fragment 4 : « GH »
- ▶ Paritätsfragment 1 : « P1 »
- ▶ Paritätsfragment 2 : « P2 »



# Fehlertoleranz

## Erasure Coding - 1 + 2 Verloren

- ▶ Fragment 1 : Paritätsfragment 1 XOR Fragment 2 = « P1 »  
XOR « CD » = « AB »
- ▶ Fragment 2 : Paritätsfragment 1 XOR Fragment 1 = « P1 »  
XOR « AB » = « CD »

Ursprüngliche Datei « ABCDEFGH »

# Fehlertoleranz

## Wiederherstellung

- ▶ Ziel : In einen korrekten Zustand zurückzukehren
- ▶ Systemausfälle erkennen und reagieren
- ▶ Verschiedenen Ebenen : Prozess, Knoten oder System

# Fehlertoleranz

## Wiederherstellungstechniken

- ▶ Checkpointing
- ▶ Nachrichtenprotokollierung
- ▶ Rollback-Recovery

# Fehlertoleranz

## Kommunikationsprotokolle

- ▶ Timeout-basierte Protokolle
- ▶ Bestätigungsbasierte Protokolle
- ▶ Protokolle für den Umgang mit Netzwerkpartitionen

# Fehlertoleranz

## Lastverteilung

- ▶ Effiziente Nutzung der Ressourcen
- ▶ Gleichen Daten und/oder Prozesse auf verschiedenen Knoten
- ▶ Statische Lastverteilung und dynamische Lastverteilung.

# Fehlertoleranz

## Anti-Entropy

- ▶ Konsistenz gewährleisten
- ▶ Zwang Repliken zu synchronisieren

# Fehlertoleranz

## Anti-Entropy-Repair

- ▶ Auch Anti-Entropy-Synchronisation
- ▶ Paare von Repliken vergleichen Daten
- ▶ Periodisch oder auf Anforderung
- ▶ Herausforderung auf skalierenden Systemen
- ▶ Nicht sofortige Konsistenz (eventual consistency)
- ▶ Konflikte (?)