Fehlertoleranz in Algorithmen für schwach-konsistente Replikation in verteilten Systemen

Felix Gärtner

Übersicht

1. Begriffe: Fehlertoleranz und Replikation
2. schwach-konsistente Replikationsalgorithmen
3. Fehlertoleranz durch Selbststabilisierung
4. selbststabilisierende Replikationsalgorithmen

Fehlertoleranz

- Herkömmliche Fehlertoleranzuntersuchungen: Korrektheit in Bezug auf Fehlermodell (*crash*, *failstop*, etc.)
- permanente vs. vorübergehende Fehler
- maskierende vs. nichtmaskierende Fehlertoleranz

Replikation (von Diensten)

Bereitstellung mehrerer identischer Kopien eines Datenelementes an verschiedenen Stellen in einem verteilten System.

+ Steigerung der Leistung (*performance*)
+ Steigerung der Verfügbarkeit (*availability*)
– erhöhter Änderungsaufwand
– Gefahr von Inkonsistenzen

starke/strenge Konsistenz $\leftrightarrow$ schwache Konsistenz

Replikation vs. Gruppenkommunikation

Gruppenkommunikation = *broadcast*/*multicast*

Replikationsverfahren $\Leftrightarrow$ Gruppenkommunikation

Drei Parameter der Zustellung (*not* =Empfang) von Nachrichten bei Gruppenkommunikation:

- Zuverlässigkeit (*delivery reliability*)
  (vollständig, zuverlässig, k-zuverlässig, bestmöglich)
- Latenz (*delivery latency*)
  (zeitbeschränkt, allmählich)
- Ordnung (*delivery ordering*)
  (keine, FIFO, kausal, total und Kombinationen)

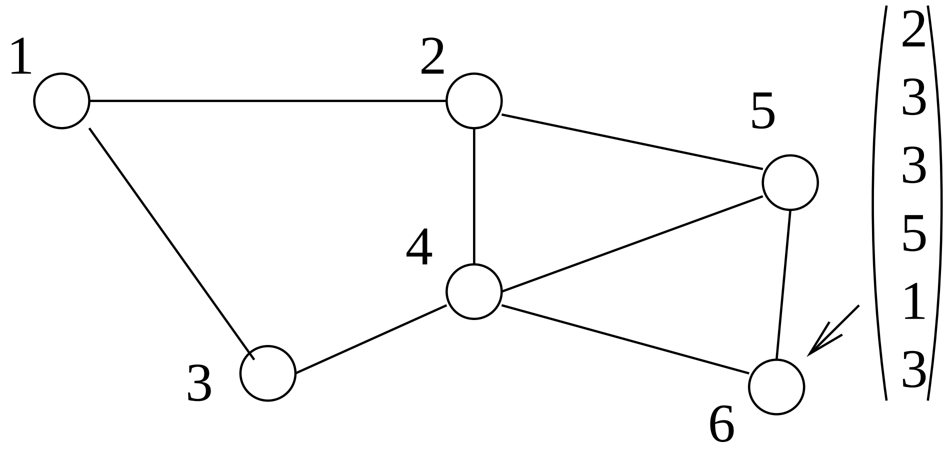
Ordnungen durch Transformationen erzeugbar.

strenge Konsistenz: vollständige, zeitbeschränkte Zustellung
(bzw. k-zuverlässig mit totaler Ordnung)

schwache Konsistenz: zuverlässige, allmähliche Zustellung mit unterschiedlichen Ordnungen

TSAE (Golding, 1992)

timestamped anti-entropy = *multicast* mit zuverlässiger, allmählicher Zustellung (+ Ordnungen)



- Nachrichten werden gepuffert (*message log*),
- jeder Knoten ist ein "Beobachter" des Systems
- periodischer Austausch der "Sichtvektoren"

⇒ Knoten weiß, was er noch nicht hat und was er aus dem Puffer löschen darf.

Analyse ("optimiertes *flooding*"):

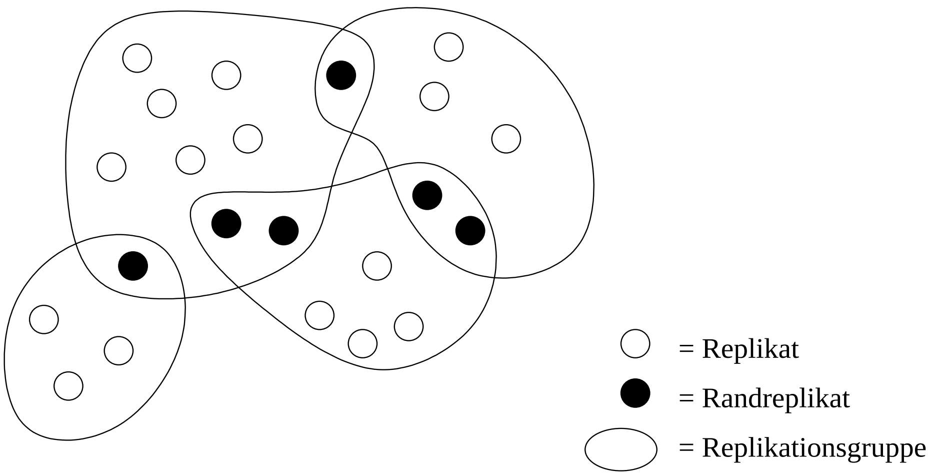
+ zuverlässige Zustellung auch bei Partitionen

± stabiler Speicher notwendig

− skalierbar? (Aufwand für Vektoruhren)

flood-d (Obraczka, 1994)

- Hierarchie von Replikationsgruppen
- Kommunikationsstruktur dynamisch anpassen



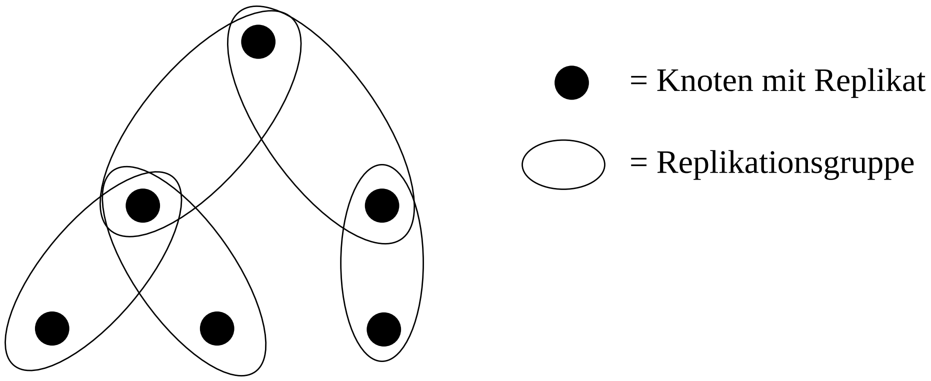
- wie $TSAE$, aber mit Randreplikaten,
- pro Gruppe sammelt einer Daten und rekonfiguriert

\+ Hierarchie $\rightarrow$ *administrative domains*

\+ Speicheraufwand $O(r)$
($r = |\text{Replikationsgruppe}|$)

\+ Optimierungsstrategien

HPP (Kumar und Adly, 1996)



- Zweiergruppen
- *flooding* über Baumstruktur (sofort und nicht periodisch)
- *stop and wait* pro Kanal
- "degeneriertes" *flood-d*

\+ Effizienz (keine redundanten Nachrichten)

– Fehlertoleranz (Baumstruktur, Grad der Fehlertoleranz vorher festlegen)

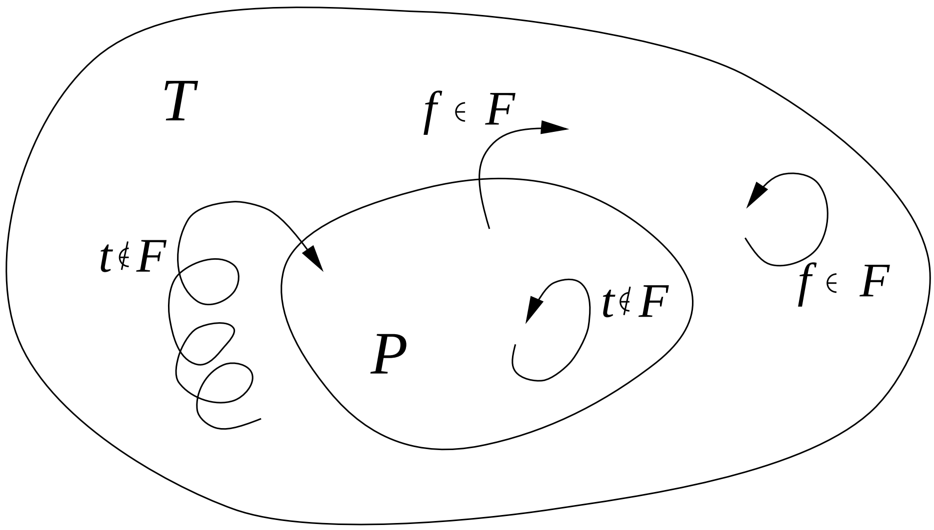
Selbststabilisierung

- Menge Z von (fehlerhaften) Systemzuständen
- Prädikat $P \subseteq Z$ beschreibt "legale" Zustände

Definition: Selbststabilisierung

Ein System stabilisiert sich selbst zu P, gdw.

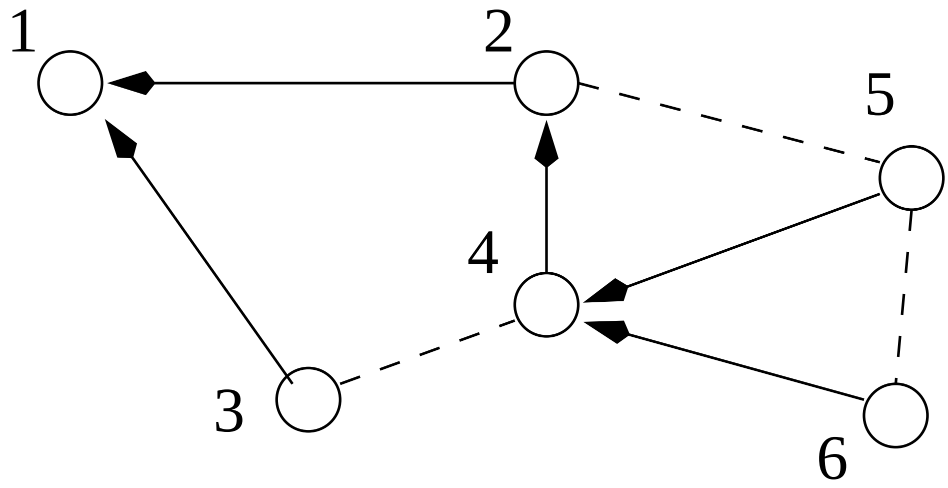
1. Gilt P, so wird P bei normaler Ausführung nicht verlassen.
2. *Jede* Ausführung enthält einen Zustand aus P.



+ toleriert beliebige vorübergehende Fehler

− nicht-maskierende Fehlertoleranz

selbststabilisierende Replikationsalgorithmen



- Spannbaum mit Wurzel
- Knoten überprüfen periodisch Replikatwert des Elternknotens
- Aktualisierung falls ungleich

$\Rightarrow$ stabilisiert sich zu Konsistenz

Erweiterungen:

1. dynamische Kommunikationstopologie: selbststabilisierenden Spannbaumalgorithmus verwenden

2. Jeder soll “senden” dürfen: Algorithmus n-fach überlagern

$\Rightarrow$ selbststabilisierender *broadcast*

Analyse der Fehlertoleranz

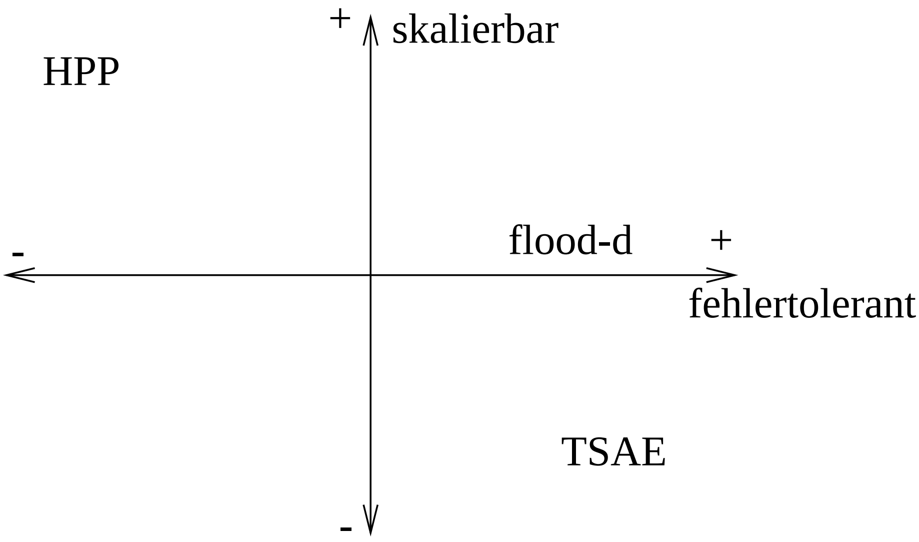
+ verträgt alle vorübergehenden Fehler

+ verträgt permanente Fehler solange verbunden

+ kann ihn als “Basisalgorithmus” für viele Replikationsverfahren verwenden (“Familie”)

– ist nicht-maskierend

Vergleich der Verfahren

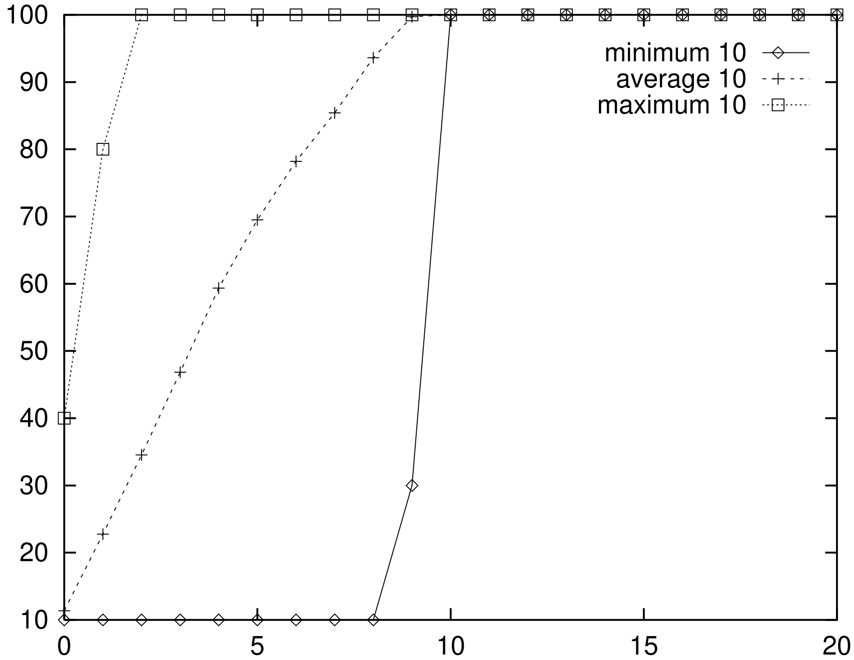


TSAE, *flood-d* und *HPP* nicht selbststabilisierend

- wegen unbeschränkten Puffern,
- wegen "Vektorzeit"

Wer toleriert schon einen *memory crash*?

Analyse der Konvergenz



$\Rightarrow$ konvergiert linear?!

Anwendungen:

- hohe Zuverlässigkeit, aber nicht auf Korrektheit bauen
- wenig Wartungsaufwand: zusätzlicher Dienst ohne Garantien, dafür kostenlos
- Zustelldienst in mobilen (feindlichen?) Umgebungen (SMS)
- Informationsdienste

Zusammenfassung

1. Begriffe: Fehlertoleranz und Replikation
2. schwach-konsistente Replikationsalgorithmen
3. Fehlertoleranz durch Selbststabilisierung
4. selbststabilisierende Replikationsalgorithmen durch selbststabilisierenden *broadcast*