

Fehlertoleranz

Motivation

Fernaufwurfsemantiken

Implementierung


Ausfallbehandlung

Transparenz



- Fehlerszenarien in verteilten Systemen (Beispiele)
 - **Unabhängige Ausfälle von Knoten**
 - Netzwerkfehler (z. B. Nachrichtenverlust, Netzwerkpartitionierung)
 - Überlast (Knoten oder Netzwerk)

→ Zuverlässige Fehlererkennung ist im Allgemeinen nicht möglich

→ Fehlertoleranzansätze sollten keine perfekte Fehlererkennung bedingen
- Herausforderungen
 - Wie kann ein Fernaufrufsystem **temporäre Netzwerkfehler** tolerieren?
 - Wie lassen sich zwischenzeitliche Knotenausfälle behandeln?
 - Wiederherstellung von Servern
 - Ausfall und Neustart von Clients
- Literatur
 -  Peter Bailis and Kyle Kingsbury
The network is reliable
ACM Queue, 12(7):20–32, 2014.



- Ideale Semantik im lokalen, fehlerfreien Fall: *Exactly-Once*
 - Einmaliger Aufruf
 - Einmalige Ausführung der aufgerufenen Prozedur / Methode
 - Einmalige Rückgabe des Ergebnisses

→ **Problem: Semantik ist im (verteilten) Fehlerfall nicht erreichbar**
- Grundlegende Techniken zur Tolerierung von Fehlern bei Fernaufrufen
 - Überwachung von Timeouts
 - Wiederholtes Senden von Nachrichten
 - Filterung von Duplikaten
- Verschiedene Aufrufsemantiken durch **Kombination der Techniken**
 - *Maybe*
 - *At-Most-Once*
 - *At-Least-Once*
 - *Last-of-Many*



- *Maybe*
 - **Einmaliges Senden** der Anfrage
 - Abbruch nach Timeout bei Ausbleiben einer Antwort
- *At-Most-Once*
 - Wiederholtes Senden der Anfrage nach Timeout, falls Antwort ausbleibt
 - **Ausführung der ersten Anfrage**, die den Server erreicht
 - Senden der ursprünglichen Antwort, falls weitere Anfragen eintreffen
- *At-Least-Once*
 - Wiederholtes Senden der Anfrage nach Timeout, falls Antwort ausbleibt
 - **Ausführung jeder Anfrage**, die den Server erreicht
 - Senden der jeweils zur aktuellen Ausführung gehörigen Antwort
- *Last-of-Many*
 - Erweiterung von *At-Least-Once*
 - Client akzeptiert nur die Antwort, die zu seiner **neuesten Anfrage** gehört



Vergleich der Fernaufrufsemantiken

■ Charakteristika

	Maybe	At-Most-Once	At-Least-Once	Last-of-Many
Timeouts	Ja	Ja	Ja	Ja
Wiederholtes Senden	Nein	Ja	Ja	Ja
Duplikatunterdrückung	Nein	Ja	Nein	Nein
Mehrfache Ausführung	Nein	Nein	Möglich	Möglich
Antwortselektion	Nein	Nein	Nein	Ja

■ Client-Wissen über die Ausführungsanzahl in verschiedenen Szenarien

	Maybe	At-Most-Once	At-Least-Once	Last-of-Many
Erfolgsfall nach einmaliger Anfrage	1	1	1	1
Erfolgsfall nach mehrmaliger Anfrage	-	1	≥ 1	≥ 1
Ausbleiben einer Antwort trotz mehrmaliger Anfrage	≤ 1	≤ 1	≥ 0	≥ 0

[Während *At-Least-Once* nach der Semantik im Erfolgsfall benannt ist, verweist *At-Most-Once* auf den Fehlerfall.]



- *At-Most-Once*
 - Duplikaterkennung erfordert **eindeutige Identifizierung** eines Aufrufs
 - Aufrufer
 - Prozedur- / Methodenaufruf
 - **Speicherung von Antworten**
 - Strategien zur Garbage-Collection des Antwortspeichers
 - Löschen der alten Antwort bei einer neuen Anfrage desselben Aufrufers
 - Verwerfen einer Antwort nach Timeout
- *At-Least-Once*
 - Zuordnung von Antwort zu Anfrage nötig → Identifizierung von Aufrufen
 - **Kein dauerhaftes Wissen** über Fernaufrufe auf Server-Seite erforderlich
- *Last-of-Many*
 - Unterscheidung von Anfragen desselben Aufrufs → **Sequenznummern**
 - Server-seitiges Wissen abhängig von Interpretation der Semantik
 - Akzeptierte Antwort gehört zu neuester Anfrage → Kein zusätzliches Wissen
 - Akzeptierte Antwort gehört zu letzter Ausführung → Speicher für Anfrage-IDs

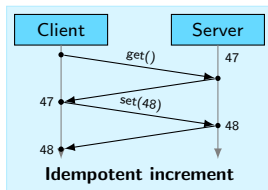
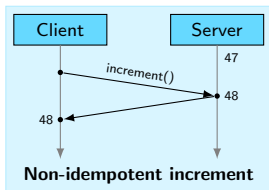


- Wiederholtes Senden von Anfragen
 - Allgemein
 - Zu beachten: **Netzwerklatenz und Ausführungsdauer** einer Methode
 - Timeout kann zwischen Versuchen variieren
 - Abbruch bei Erreichen einer bestimmten Anzahl von Versuchen
 - **Protokollabhängige Semantik**
 - UDP: Timeout als Schutz gegen verlorene Nachrichten
 - TCP: Timeout als Schutz gegen fehlgeschlagene Verbindungen
- Risiken bei **ungünstiger Timeout-Wahl**
 - Senden von Anfragen
 - Zu kurz: Livelock (*Last-of-Many*) bzw. unnötiges Senden von Nachrichten
 - Zu lang: Erhöhte Antwortzeit bei Nachrichtenverlust
 - Garbage-Collection bei *At-Most-Once*
 - Zu kurz: Mehrfachausführung einer Anfrage → Verletzung der Garantien
 - Zu lang: Verlust der Verfügbarkeit aufgrund eines vollen Antwortspeichers



Konsistenzwahrung bei Aufrufwiederholungen

- **Idempotenz** [„idem“ (lat.): dasselbe.]
 - Wiederholung eines Aufrufs hat **denselben Effekt** wie einmaliger Aufruf
 - Annahme: Isolierter Aufruf → Keine Überlagerung mit anderen Aufrufen
- Ansätze im Kontext von Fernaufrufen
 - Erkennung und Unterdrückung von Aufrufwiederholungen im Skeleton
 - Umsetzung auf Anwendungsebene bei **zustandslosen Skeletons**
 - Keine Zustandsinformationen über die Client-Interaktion im Skeleton
 - Mehrfachausführung führt zu identischen Ergebnissen bzw. Zuständen
- Beispiel: Nichtidempotente vs. idempotente Implementierung



- Anforderungen bei Ausfall und Neustart eines Servers
 - Zustandsbehaftete Dienste
 - **Kein Verlust bestätigten Applikationszustands**
 - Beispiel: Vom Dienst beantwortete Schreibaufrufe
 - Sicherstellung durch die Anwendung
 - Fernaufrufsystem
 - **Keine Verletzung der Fernaufrufsemantiken**
 - Beispiel: Keine Mehrfachausführung bei *At-Most-Once*
 - Mechanismus zur Zustandswiederherstellung erforderlich
- Absicherung durch **Protokollierung essentieller Informationen**
 - Persistente Speicherung (Beispiele)
 - Log-Datei(en) auf lokaler Festplatte
 - Nutzung eines separaten (fehlertoleranten) Systems
 - Einspielen des Protokolls beim Neustart
 - Garbage-Collection auf Basis von Sicherungspunkten




- Vorgehensweise auf Server-Seite
 1. Erhalt der Anfrage
 2. Protokollierung der **eindeutigen Aufrufkennung**
 3. Bearbeitung der Anfrage in der Anwendung
 4. Protokollierung der **Antwort**
 5. Senden der Antwort
- Verhalten des Servers bei Erhalt einer Anfrage nach Neustart

Aufruf	Antwort	Szenario	Reaktion
Unbekannt	Unbekannt	Neue Anfrage oder Ausfall vor Schritt 2	Ausführung der Anfrage Senden der Antwort
Bekannt	Unbekannt	Ausfall nach Schritt 2 aber vor Schritt 4	Senden einer Fehlermeldung
Bekannt	Bekannt	Ausfall nach Schritt 4	Senden der Antwort

⇒ Wiederherstellung kann zu **nichttolerierbaren Situationen** führen




- Problemszenario: Verwaiste Fernaufrufe (*Orphans*)
 - Start einer **(zeit)aufwändigen Operation** per Fernaufruf
 - Ausfall des aufrufenden Clients vor Erhalt der Antwort→ Ziel: Abbruch des Fernaufrufs
- Annahme: Abbruch einer Operation führt nicht zu Inkonsistenzen
 - Geordnete Rückgabe von Locks
 - Rückgängigmachung von Zustandsänderungen
 - ...
- Erhöhte Komplexität bei **Fernaufрукetten**
 - Bearbeitung eines Fernaufrufs erfordert selbst wiederum Fernaufrufe
 - Beispiel: Aufspaltung großer Aufgaben in Teilaufgaben
- Literatur
 -  Bruce Jay Nelson
Remote procedure call
Dissertation, Carnegie-Mellon University, CMU-CS-81-119, 1981.



- Explizite Terminierung durch den Client (*Extermination*)
 - Protokollierung von Fernaufrufen durch den Stub
 - **Abbruchanfragen** für nicht mehr benötigte Fernaufrufe nach Neustart
 - Nachteil: Persistente Schreibaufrufe vor und nach jedem Fernaufruf
- Einsatz systemweiter *Epochen* (*Reincarnation*)
 - Nachrichten tragen Epochenkennung (= Zählerwert)
 - Beginn und **Bekanntgabe einer neuen Epoche** bei Neustart
 - Wechsel in neue Epoche: Abbruch aller Operationen früherer Epochen
 - Variante: *Gentle Reincarnation*
 - Nachfrage beim Client
 - Abbruch einer Operation, falls kein Widerspruch des Clients vorliegt
- Absicherung durch Timeouts (*Expiration*)
 - Setzen von Fristen für Fernaufrufe
 - Bei Bedarf: Schrittweise Fristverlängerung durch den Client
 - Beendigung eines Aufrufs bei **Ablauf der Frist**



- Generelles Problem: Angleichung der Sichten verschiedener Knoten
 - Globale Entscheidungen erfordern **konsistente lokale Sichten** (Beispiele)
 - Soll ein Fernaufruf abgebrochen werden?
 - *At-Most-Once*: Kann eine Antwort verworfen werden?
 - Welcher Knoten übernimmt aktuell die Anführerrolle im System?
 - Abstimmung lokaler Sichten durch Interaktion ist nicht immer möglich
- *Lease*
 - Zeitliche Begrenzung für von Knoten getätigte Aussagen
 - Lease-Verlängerung benötigt **erfolgreiche aktive Handlung**
 - Ablauf eines Lease: Setzen der lokalen Sicht auf einen vordefinierten Wert
 - Voraussetzung: Hinreichend genaue und ähnlich schnelle Uhren
- Literatur
 -  Cary G. Gray and David R. Cheriton
Leases: An efficient fault-tolerant mechanism for distributed file cache consistency
Proceedings of the 12nd Symposium on Operating Systems Principles (SOSP '89),
S. 202–210, 1989.



- Problem
 - Nicht alle Fehlersituationen lassen sich im Fernaufrufsystem tolerieren
 - Beispiele
 - Permanente Netzwerkfehler
 - Knotenausfälle
 - Inkompatible Stubs und Skeletons
- **Abbruch des Fernaufrufs** erforderlich
- *Remote-Exception*
 - **Signalisierung** im Fernaufruf begründeter Ausnahmesituationen
 - Vorgehensweise abhängig vom Ort des Auftretens
 - Stub: Rückkehr aus dem Fernaufruf mittels Exception
 - Skeleton: Weiterleitung der Exception zum Stub, danach an die Anwendung
- Reaktion der Anwendung (Beispiele)
 - Rückgriff auf **weiterführende Fehlertoleranzmechanismen**
 - Benachrichtigung des Nutzers bzw. Administrators



- Überblick über bisher behandelte Maßnahmen
 - Stub und Skeleton
 - **Stellvertreter für Server bzw. Client**
 - Abfangen lokaler Methodenaufrufe
 - Abbildung auf Nachrichtenaustausch
 - Bereitstellung von Ortstransparenz
 - Automatisierte Generierung von Stubs und Skeletons
 - **Tolerierung von Nachrichtenverlusten** mittels Fernaufrufsemantiken
 - Problem
 - Existenz im Fernaufruf bedingter Fehler, die eine Signalisierung erfordern
 - Anwendung muss auf solche Ausnahmesituationen vorbereitet sein
- ⇒ Erkenntnis
- Das komplexe Fehlermodell verteilter Systeme macht es unmöglich, Fernaufrufe in jedem Fall vollständig transparent zu realisieren**

