Verteilte Systeme

14. Fehlertoleranz

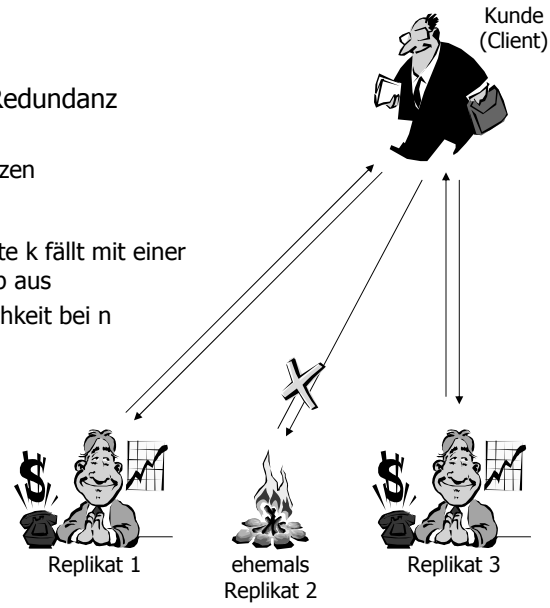
Warum?

- Zunehmende Durchdringung
  - Menschenleben hängen von kritischen Computersystemen ab
  - Unternehmen hängen von der Verfügbarkeit der IT ab
- $\Rightarrow$ Hohe Verfügbarkeit (HA)
- Kompensation von Hardware- und ggf. Softwarefehlern
- Redundanzen

Motivation

- Verteiltes System = Redundanz
  - Rechnern
  - Kommunikationsnetzen
- Grundidee
  - Einzelne Komponente k fällt mit einer Wahrscheinlichkeit p aus
  - Ausfallwahrscheinlichkeit bei n Replikaten von k:

$$p^n < p$$



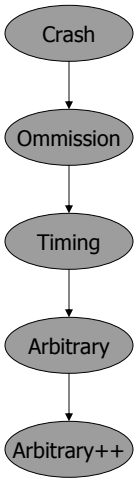
Systemsoftware II, Winter 2002/03 Folie 14.3

High Availability

- 99% HA
  - 3 Tage und 15 Stunden Ausfall pro Jahr
  - 26 Tage und 12 Stunden Ausfall pro 10 Jahre
- 99.9% HA
  - 8 Stunden und 45 Minuten Ausfall pro Jahr
  - 3 Tage und 15 Stunden Ausfall pro 10 Jahre
- 99.99% HA
  - 53 Minuten Ausfall pro Jahr
  - 8 Stunden und 45 Minuten Ausfall pro 10 Jahre
- 99.999% HA
  - 5 Minuten Ausfall pro Jahr
  - 53 Minuten Ausfall pro 10 Jahre
- und so weiter

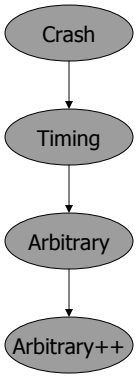
Systemsoftware II, Winter 2002/03 Folie 14.4

Fehlermodelle



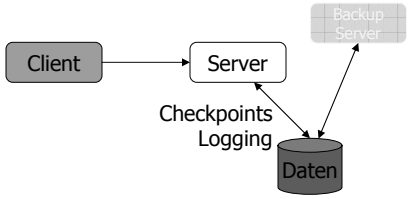
- Welche Fehler soll eine Anwendung tolerieren?
  - Maskieren von Fehlern
- Aufeinander aufbauende Fehlerklassen
  - *Crash*
    - Sofortiger, dauerhafter und beobachtbarer Stop (Fail-and-Stop oder auch Fail-Silent)
    - Grad der Amnesie
  - *Ommission*
    - Auslassung
  - *Timing*
    - Zu früh (Echtzeitfehler)
    - Zu spät (Performance-Fehler)
  - *Arbitrary*
    - Beliebige Fehler, aber ehrlich gemeint
  - *Arbitrary with Message Authentication*
    - Beliebige Fehler inklusive absichtlicher Fälschungen

Modellvariante

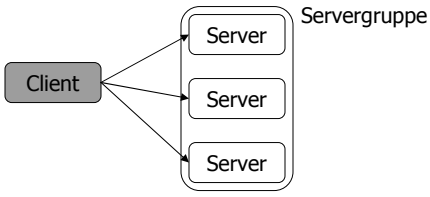


- Ommission wird durch Wiederholungen beliebig unwahrscheinlich
- Übergang zu Timing-Fehlern
- Timing-Fehler sind zeitlich begrenzt
- Entdeckung in endlicher Zeit (Timeout)

Ansätze



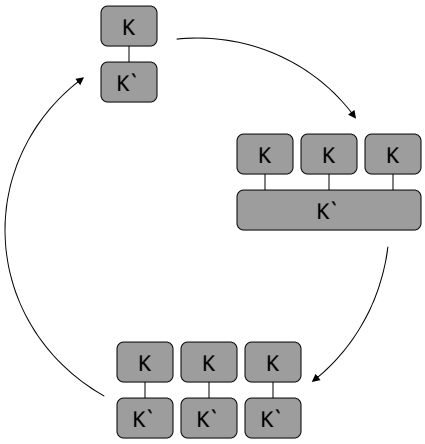
- Passive Redundanz
- Replikation von
  - Daten
- Vorteil: Einfache Realisierung
- Nachteil: Nicht alle Fehlerklassen maskierbar
  - Welche nicht?



- Aktive Redundanz
- Replikation von
  - Daten
  - Kontrollfluß
- Vorteil: Alle Fehlerklassen im Prinzip maskierbar
- Nachteil: Aufwendige Hardware- und Software-Realisierung

Systemsoftware II, Winter 2002/03 Folie 14.7

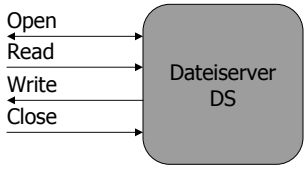
Mathematischer Hintergrund



- Einzelne Komponente hat Ausfallwahrscheinlichkeit p
- Ziel: N-fache Replikation der mit Ausfallwahrscheinlichkeit p^n
- Ausfälle müssen unabhängige Ereignisse sein
  - Andere Komponenten
  - Systemsoftware
  - Hardware
- Zuverlässigkeit der Komponente hängt von vielen anderen Komponenten im System ab
- Gesamtsystem kann nicht zuverlässiger als die unzuverlässigste Komponente werden

Systemsoftware II, Winter 2002/03 Folie 14.8

Beispiel



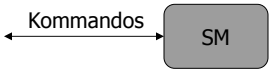
- Einfacher, nicht fehlertoleranter Dateiserver DS
- Maskierung maximal eines Ausfalls
- Realisierung
  - Crash?
  - Ommission?
  - Timing?
  - Arbitrary?
  - Arbitrary mit Nachrichtenauthentifizierung?

Wieviel Redundanz ist notwendig

- Abhängig von
  - Maskierte Fehlerklasse
  - Tolerierbare Anzahl an gleichzeitigen Ausfällen
- K-Zuverlässigkeit
  - Servergruppe toleriert den gleichzeitigen Ausfall von k Mitgliedern
- Replikationsgrad k+1
  - Crash, Ommission und Timing
  - Schnellste gewinnt
- Replikationsgrad 2k+1
  - Arbitrary
  - Mehrheitsentscheid
- Replikationsgrad 3k+1
  - Arbitrary with Message Authentication

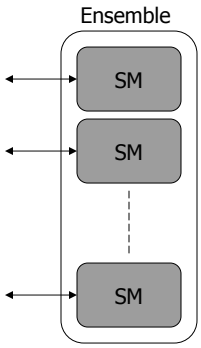
Aktive Redundanz: State-Machine-Approach

- Zustandsmaschine
  - Zustandsvariablen
  - Zustandsverändernde Kommandos: `a=sm.k(e)`
- Determinismus und Atomarität
  - a hängt nur vom initialen Zustand und der Abarbeitungsreihenfolge vorangegangener Kommandos ab
- Funktion der Zustandsmaschine unabhängig von
  - Zeit
  - Anderen Systemaktivitäten
- Keine Seiteneffekte
- Sind Zustandsmaschinen leicht realisierbar?



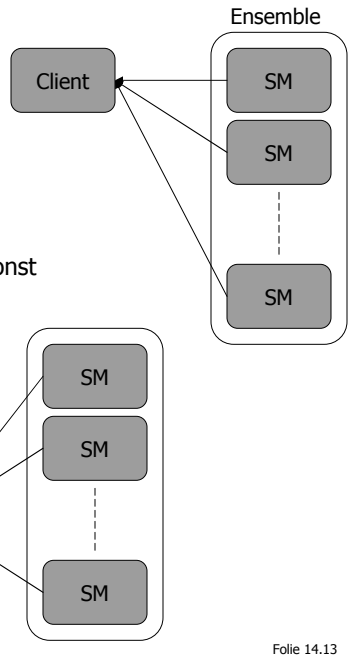
Ein Ensemble

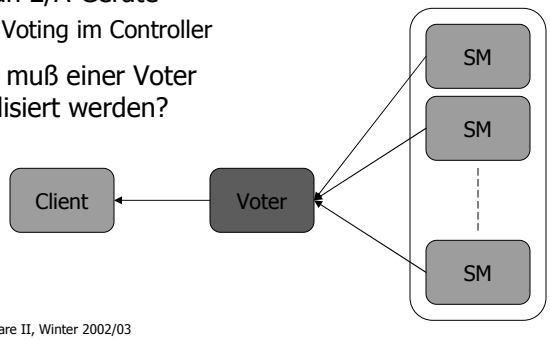
- Replizierte State-Machines
- Aktive Redundanz bei Gleichtaktung
  - Gleicher initialer Zustand
  - Kommandos in der gleichen Reihenfolge
- Replikatkoordination
  - Alle nicht-fehlerhaften Zustandsmaschinen erhalten alle Kommandos
  - Ausführung der Kommandos in der gleichen Reihenfolge
- Realisierungsvarianten
  - Eigene Implementierung
  - Geordneter Multicast und ...?
    - Welcher Ordnungsgrad ist minimal notwendig?



Ausgaben eines Emsembles

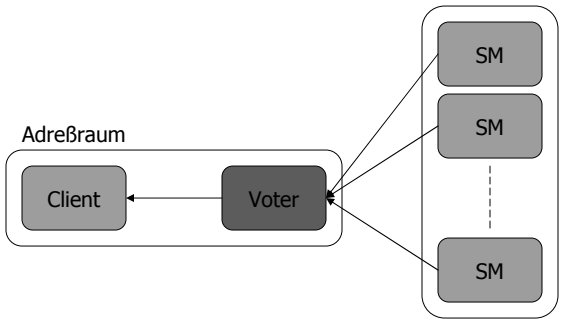
- ... an andere Komponenten
  - Erwartet wird nur ein Reply
  - Voting
    - 1. Reply bis Timing
    - Nach k+1 identischen Replies sonst
- ... an E/A-Geräte
  - Voting im Controller
- Wo muß einer Voter realisiert werden?





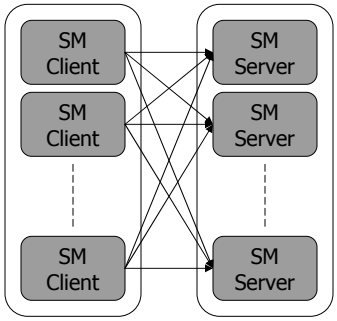
Lokalisierung Voter

- Warum nicht als eigenständiger Prozeß?
- Voter ist Teil einer Laufzeitbibliothek im Client



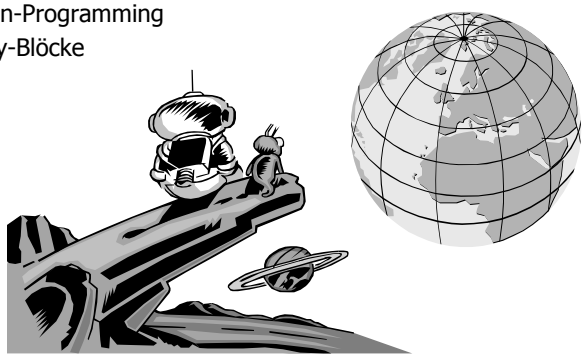
Unzuverlässige Clients

- Fehlerhafte Clients können k-zuverlässiges Ensemble stören
  - Trotzdem alle Aufträge in gleicher Reihenfolge bearbeitet werden
  - Vergleichbar "Denial of Service"-Attacken
- K-zuverlässiges Ensemble $\not\Rightarrow$ k-zuverlässiges System
- Lösungsansätze
  - Defensive Programmierung
  - Clients ebenfalls replizieren
    - Zu maskierende Fehlerklasse?
    - Plazierung des serverseitigen Voters?



Wirklich Fehlerklasse "Arbitrary"?

- Maskierung von Softwarefehlern
- Einfache Replizierung einer State-Machine reicht nicht
- Lösungsansätze
  - N-Version-Programming
  - Recovery-Blöcke
- Aufwand



Rekonfigurierung

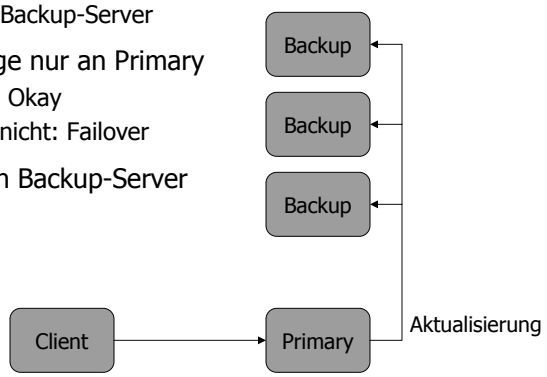
- Reaktion auf ausgefallene Replikate
- Fehlerklasse Crash bis Timing
  - Erzeugung eines Ersatzservers
  - Einphasen mit dem Ensemble
    - Selbststabilisierende Replikate (vgl. Dijkstra 1974)
    - Abgleich über korrektes Ensemblemitglied
- Fehlerklasse Arbitrary
  - Kein Ersatz integrierbar
  - Zuverlässigkeitsgrad wird kleiner
  - Aufwendige und kontrollierte Neuinitialisierung bei zu geringem Zuverlässigkeitsgrad

Zusammenfassung State-Machine-Approach

- Ein Ansatz für aktive Redundanz
  - Einschränkungen an die Replikat-Realisierung
    - Determinismus
    - Atomarität
  - Total geordneter Multicast
  - Anwendungsunabhängige Voter
    - Mehrheitsentscheid: Bitweiser Vergleich der Replies
- Materialschlacht
- Zuverlässigkeitsgrad k>1 selten

Passive Redundanz: Primary-Backup-Approach

- Ein Primary-Server
- Backup-Server im "Hintergrund"
  - Aktualisierung der Backup-Server
- Client sendet Aufträge nur an Primary
  - Primary antwortet: Okay
  - Primary antwortet nicht: Failover
- Im Fehlerfall wird ein Backup-Server zum neuen Primary



Aktualisierungsfrequenz

- Hot Standby
  - Jedes Kommando wird nachgezogen
- Warm Standby
  - Periodische Übernahme des Primaryzustands
- Cold Standby
  - Übernahme des Primaryzustands nach Ausfall
- Unterschiede in der Reaktionszeit
- Election bei Primaryausfall

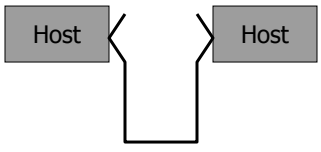
Passive Redundanz in der Praxis

- Hardware-Redundanz
  - Kein SPOF
  - Rechner, Persistenter Speicher, Netzanschlüsse
- Gegenseitige Überwachung
  - Heartbeat-Protokolle
- Freundliche Übernahme (Takeover)
  - … des Speichersubsystems
  - … der IP-Adresse
  - … der MAC-Adresse

Speicher-Redundanz

- Günstige Lösungen
  - SCSI-Bus mit dualem Zugang
  - Abstand bei Differential SCSI: ~25m
  - STONITH-Problem
- RAID-Speichersysteme
- Spezielle Speichersysteme mit Mehrfachanschlüssen
  - SAN oder NAS
- Voll redundante Server
  - Datenreplikation

Shoot the other node in the head



Netz-Redundanz

- Idealerweise zwei Netzkarten (z.B. Ethernet)
  - Primärer Netzanschluß
  - Sekundärer passiver Netzanschluß
- Zwei Subnetze
  - Rechner zur Not über Alternativanschluß erreichbar
- Varianten
  - Übernahme der IP-Adresse (IPAT, Achtung: ARP-Caches)
  - Übernahme der MAC-Adresse
    - So eine Karte hätte ich auch gerne ☺
  - Existierende TCP-Verbindungen brechen

Heartbeat-Protokolle

- Periodische Lebenszeichen aller Mitglieder
- Multicast- und Broadcast-basierte Verfahren
  - Problematik der Broadcast-Stürme
  - Vergleichsweise geringe Frequenz
- Aufgesetzte Agreement-Protokolle
  - Wann gilt Teilnehmer als ausgefallen?
  - Problematik Netzpartitionen
  - Fairneß nicht notwendig
  - Membership meist Huckepack mit Heartbeat
- Sicherheits- und Schutzaspekte

Anforderungen

- Breite Unterstützung unterschiedlicher Protokolle
  - Ethernet, Serielle Schnittstelle, SCSI, …
- Maximaler Ausgangsgrad bei dedizierten Leitungen
  - Meldung von Leitungsfehlern auch bei redundanten Wegen
- Broadcasts im Heartbeat-Netz (Subnetz)
  - Senden an alle Cluster-Mitglieder
- Integration zuverlässiger Multicast-Nachrichten
  - Aufgesetzte Membership- und Agreement-Protokolle

Heartbeat über V.24 o.ä.

- Best Practice
- Vorteile
  - Einfache und zuverlässige Hardware
  - Unabhängigkeit von Fehlern und Ausfällen im IP-Stack
  - Deterministische Protokolleigenschaften
  - Stecker können festgeschraubt werden ☺
- Nachteile
  - Eigener Protokollstack
  - Redundante Netzkarte weiterhin sinnvoll
  - Skalierbarkeit (maximal 30 Knoten in einem seriellen Ring organisierbar)

Takeover-Varianten (1)

- Idle Standby
  - Klassisch: Standby Node mit höchster Priorität übernimmt
  - Kurze Unterbrechung bei Eintritt eines noch-prioren Rechners
  - Standby Node kann schwächer sein
- Rotating Standby
  - FCFS-Verfahren, zweiter ist Standby
  - Gleichstarke Knoten
- Simple Failover
  - Standby bearbeitet unkritische Tasks
  - Übernimmt im Fehlerfall
  - Rückgabe nach Reintegration

Takeover-Varianten (2)

- Mutual Takeover
  - Kritische Last auf zwei und mehr Knoten verteilt
  - Jeweils Idle Standby für den anderen
  - Multiple Takeover: Lastaufteilung auf mehrere Knoten
- Concurrent Access
  - Aktive Redundanz

Weitere Problembereiche

- Cluster Partitionierungen
  - Split-Brain Syndrom
  - Eher von wissenschaftlichem Interesse
- HA = Lange Laufzeit
  - Aktualisierung und Ergänzung der Software im laufenden Betrieb
- Schutz und Sicherheit
  - Hochverfügbare Rechner werden wohl auch hochinteressant sein (DoS, Replay der Heartbeats, …)
  - Schutz durch Firewalls, Intrusion Detection, …