Verteilte Systeme

Oktober - November 2023

7. Vorlesung – 09.11.2023

Kurs: TINF21AI1

Dozent: Tobias Schmitt, M.Eng.

Kontakt: d228143@
student.dhbw-mannheim.de

Wiederholungsfragen

- Welche Aspekte aus der Vorlesung kann man für die Konzeption eines Bot-Netzes verwenden?

- Welche Gründe und Gegenindikationen kann es für Replikation geben?

- Welche Schwierigkeiten bestehen bei der Realisierung von Konsistenz?
- Was verstehen Sie unter dem Begriff von „Conit“?
- Was verstehen Sie unter „Eventual Consistency“?
- Was ist clientzentrierte Konsistenz?
- Kennen Sie Konsistenzmodelle im Rahmen der clientzentrierten Konsistenz?

Themenüberblick

- **Konsistenz und Replikation**
  - **Replikationsverwaltung**
  - Konsistenzprotokolle
- Fehlertoleranz

Replikationsverwaltung

- Um welche Kernfragen könnte sich bei der Replikationsverwaltung handeln?
- Nach welchen Kriterien könnte man Replikate verteilen / anlegen?
- Warum könnte man eine Unterscheidung zwischen serverinitiierter und clientinitiierter Replikation vornehmen? Was bedeutet das?
- Wer initiiert die Aktualisierung eines Replikates?
- Wie werden Inhalte / Aktualisierungen verteilt? Welche Arten sind hier denkbar?

Replikationsverwaltung

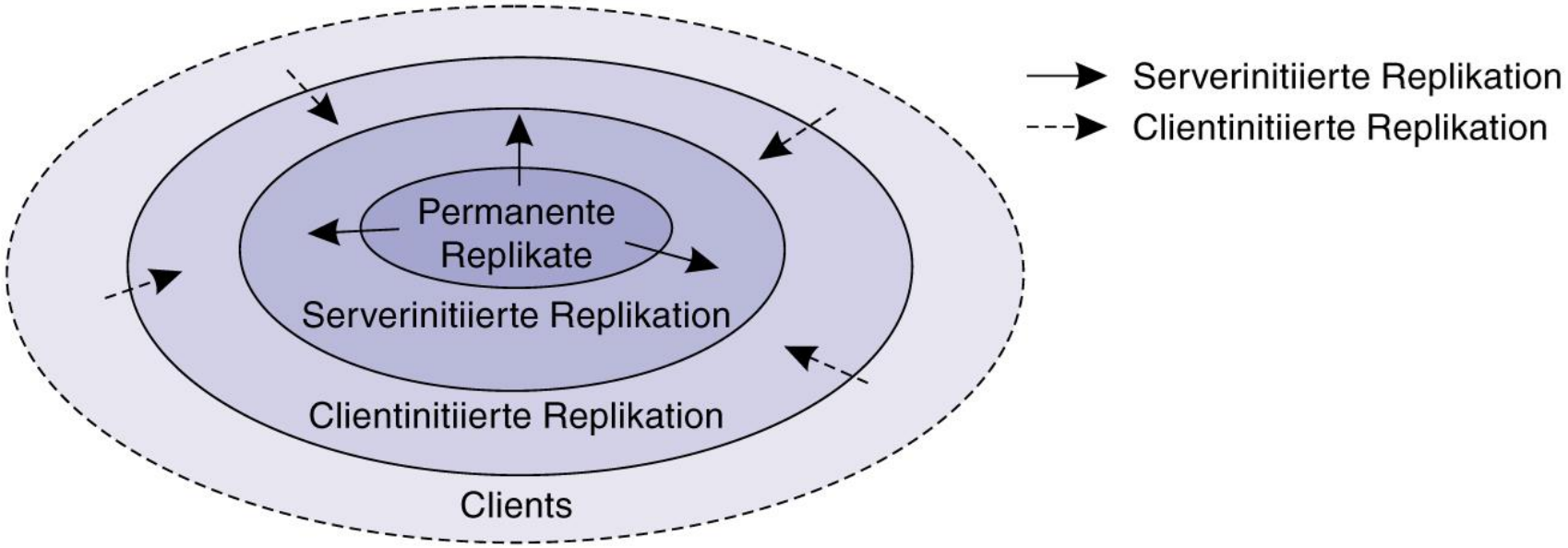
- Kernfragen:
  - Wo, wann und von wem sollen Replikate platziert werden?
- Teilaufgaben:
  - Platzierung von Replikatservern
  - Platzierung von Inhalten

Replikationsverwaltung

- Platzierung von Replikatservern
  - Achtung: Häufig verwaltungstechnische und kommerzielle Frage als Optimierungsproblem
  - Betrachtungsmöglichkeiten:
    - Geografisch (clientorientiert) – Berücksichtigung der räumlichen Entfernung
    - Topologisch – Betrachtung autonomer Systeme (System mit gleichem Routing-Protokoll; Teilnetze)
    - Nutzungsorientiert – Einteilung in Regionen mit Knoten, die auf dieselben Inhalte zurückgreifen

Replikationsverwaltung

- Replikation und Platzierung von Inhalten
  - Unterscheidung
    - Permanente Replikate
    - Serverinitiierte Replikation
    - Clientinitiierte Replikation



Replikationsverwaltung

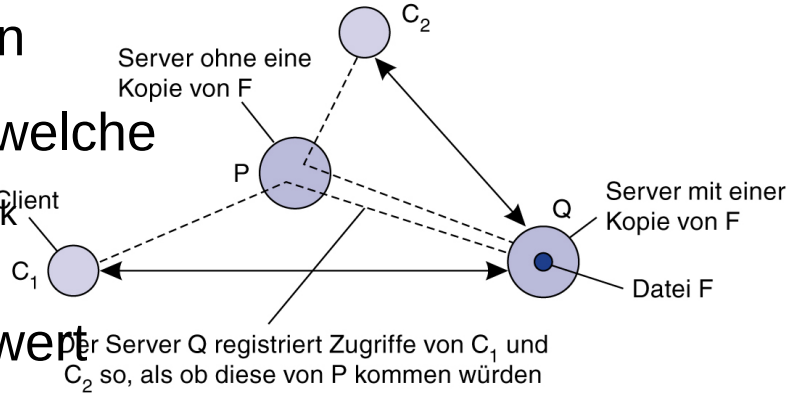
- Permanente Replikate
  - Statische Methode
  - Unterscheidungen
    - Replikate an eine Standort
      - Weiterleitung der Anforderungen nach einer Round-Robin-Strategie
    - Replikate geografische verteilt
      - alias Spiegelung (Mirroring)
      - Beispiel: Clients können aus Liste Mirror-Webseite wählen

Replikationsverwaltung

- Serverinitiierte Replikation
  - Erzeugung temporärer Replikate in Regionen aus denen Anforderungen kommen
    - z.B. Ticketverkauf eines K-Pop-Konzertes in Dtl. über koreanische Webseite
  - Einrichtung auf Initiative des Datenspeichers
  - Grund: Leistungsverbesserung
  - Annahme: Wissen über alle Server Q_i des Webhosting-Dienstes
  - **Frage:** Wie kann man bestimmen, auf welchem Server Q_i die Daten (z.B. einer Webseite) am besten aufgehoben sind?

Replikationsverwaltung

- Serverinitiierte Replikation
  - **Frage:** Wie kann man bestimmen, auf welchem Server Q_i die Daten (z.B. einer Webseite) am besten aufgehoben sind?
  - Lösungsansatz:
    - Ansatz: Überwachung der Zugriffszahlen
    - Zugriffszählung N_{ik} eine Datei F auf Q_i, welche über einen „nächstgelegenen“ Server Q_k hätte laufen können
    - Zugriffszählung N_{ik} > Replikationsgrenzwert
      - Replikation empfehlenswert
    - Zugriffszählung N_{ii} < Löschgrenzwert
      - Löschung, solange noch ein weiteres Replikat
    - Zugriffszählung N_{ii} > 2 * Summe über alle N_{kk} außer Server Q_i
      - Ggf. Migrationsempfehlung



Replikationsverwaltung

- Clientinitiierte Replikation
  - Bezeichnung: (clientseitiger) Cache
  - Hintergrund: Großteil der Zugriffe ist lesend
  - Cache auf selben Computer oder im selben LAN
    (größer angelegte Caches auch möglich)
  - Vorhaltung:
    - Auf bestimmte Zeit
    - Berücksichtigung von Cache-Treffer (Cache-Hits)
  - Datenspeicher i.A. nicht verantwortlich Daten im Cache konsistent zu halten

Replikationsverwaltung
Verteilen von Inhalten

- Ziel: Betrachtung hinsichtlich der Weiterleitung von (aktualisierten) Inhalten
- Erläuterungen zur Benachrichtigungsweiterleitung
  - Benachrichtigung über Invalidierungsprotokolle
    - Nutzung einer geringen Netzwerkbandbreite
    - Nur Infos über Ungültigkeit von Daten
    - Anwendbarkeit: Wenig Leseoperationen im Vergleich zu Aktualisierungen

Replikationsverwaltung
Verteilen von Inhalten

- Ziel: Betrachtung hinsichtlich der Weiterleitung von (aktualisierten) Inhalten
- Erläuterungen zur Benachrichtigungsweiterleitung
  - Benachrichtigung über Invalidierungsprotokolle
    - Nutzung einer geringen Netzwerkbandbreite
    - Nur Infos über Ungültigkeit von Daten
    - Anwendbarkeit: Wenig Leseoperationen im Vergleich zu Aktualisierungen

Replikationsverwaltung
Verteilen von Inhalten

- Erläuterung zur Übertragung der Daten
  - Anwendbarkeit, wenn viel mehr Leseoperationen als Aktualisierungen
  - Bündelungen möglich, zwecks Einsparung von Bandbreite

- Erläuterung zur Weiterleitung der Aktualisierungoperationen
  - Bezeichnung: aktive Replikation
  - Ziel: Einsparung von Bandbreite
  - Achtung: Aktualisierungsoperationen können unterschiedlich komplex ausfallen → ggf. mehr Rechenleistung beim Replikat nötig

Replikationsverwaltung
Verteilen von Inhalten

- Frage: Aktualisierung über Pull und Push?

- Push-basierter Ansatz = serverbasierte Protokolle
  - Einsatz meist bei dauerhaften oder serverinitiierten Replikaten
  - Ziel: hoher Konsistenzgrad
- Pull-basierter Ansatz = clientbasierte Protokolle
  - Einsatz z.B. bei Webcaches
  - Effizient, wenn Verhältnis von Lesezugriffen zu Aktualisierungen gering ist
  - Einsatz von Polling

Replikationsverwaltung
Verteilen von Inhalten

- Vergleich zwischen Push-basierten und Pull-basierten Ansätzen
  - Annahme: 1 Server und mehrere Clients

Thema	Push-basiert	Pull-basiert
Zustand auf dem Server	Auflistung der Client-Replikate und Clientcaches	Keine
Gesendete Nachrichten	Aktualisieren (sowie später möglicherweise Abrufen der Aktualisierung)	Ständiges Abfragen und Aktualisieren
Antwortzeit für den Client	Unmittelbar (oder Zeitaufwand für den Abruf der Aktualisierung)	Zeitaufwand für den Abruf der Aktualisierung

Replikationsverwaltung
Verteilen von Inhalten

- Hybride Form der Aktualisierungsweiterleitung
  - Basis: Leases (Verleihen)
  - Eine Lease ist eine Zusage eines Servers, dass er dem Client eine Zeit lang Aktualisierungen bereitstellt. Danach muss der Client beim Server Aktualisierungen abfragen.
  - Oder kurz: Push-Methode vom Server, wenn Lease vorhanden, ansonsten Pull-Methode vom Client
  - Lease-Laufzeit-Kriterien
    - Alter: längere Leases werden gewährt, wenn Objekt sich über längeren Zeitraum sich nicht verändert hat
    - Zugriffshäufigkeit: Je häufiger die Zugriffe, desto länger die Leases.
    - Zustand und Speicherplatz: Wenn Server überlastet, Verkürzung der Laufzeit.
      - Resultat: Weniger Clients zum Überwachen
      - Effizients abhänig von der Anzahl der Aktualisierungen

Replikationsverwaltung
Verteilen von Inhalten

- Fragestellung hinsichtlich Verwendung von Unicast und Multicast

- Unicast eher bei Pull-basiertem Ansatz
  - Da ein Client bei dem Server anfragt.
- Multicast eher bei Push-basiertem Ansatz

- Hinweis: Multicast häufig billiger.
  - Auf LAN-Ebene sogar Hardwareunterstützung

Themenüberblick

- **Konsistenz und Replikation**
  - Replikationsverwaltung
  - **Konsistenzprotokolle**
- Fehlertoleranz

Konsistenzprotokolle

- Beschreibung der Implementierung eines Konsistenzmodells
- Arten von Konsistenzprotokollen
  - Urbildbasierte Protokolle
    - Existenz eines ausgezeichneten Replikates als Startpunkt für Aktualisierungen
  - Nicht-Urbildbasierte Protokolle
    - Aktualisierungen können bei beliebigen Replikaten beginnen

Konsistenzprotokolle

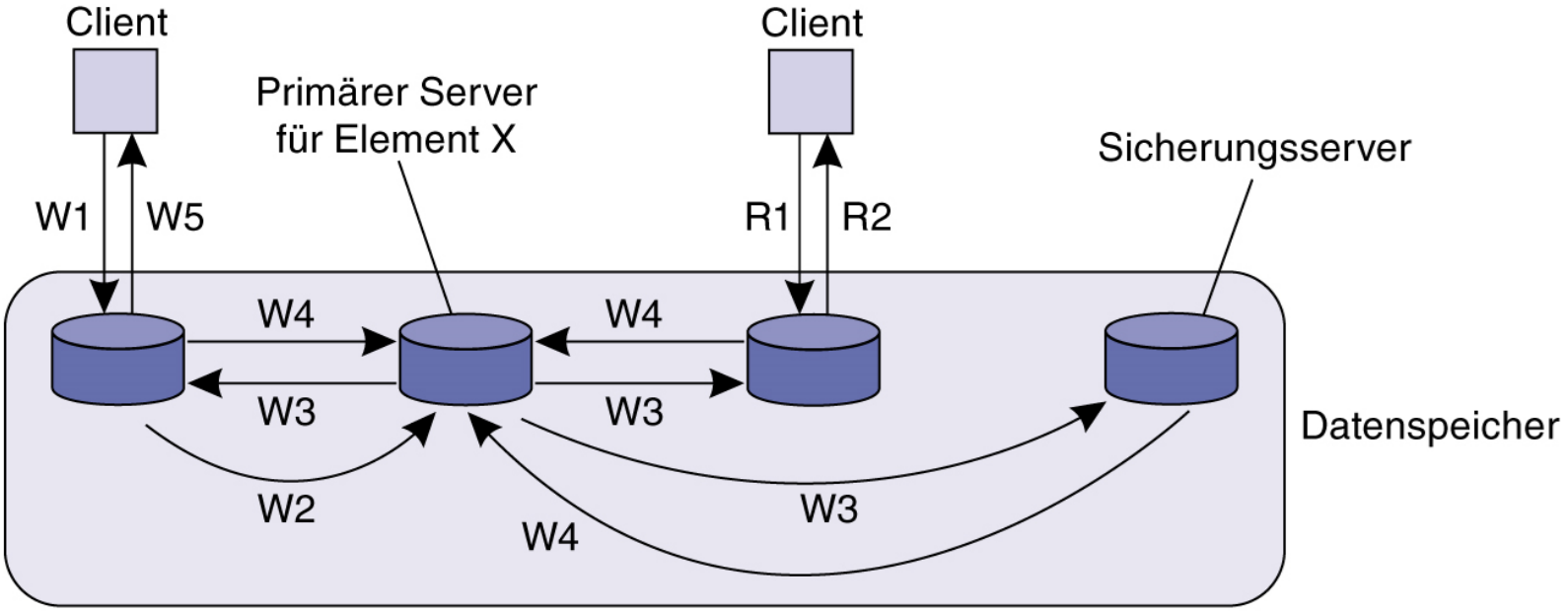
Urbildbasierte Protokolle (Primary-Based Protocols)

- Protokolle für entfernte Schreibvorgänge
  - Verfahren:
    - Datenelemente haben für sie verantwortliche Server
    - Schreiboperation an Datenelement *x* wird an entsprechenden primären Server für *x* weitergeleitet
    - Primärer Server führt Schreiboperation aus und leitet diese anschließend Backupserver weiter
    - Nach Rückmeldung von den Backupservern → Rückmeldung an den Ausgangsprozess
  - Realisierung als sperrende Operation:
    - Vorteil: Client-Prozess weiß, dass Aktualisierungsprozess von mehreren Servern abgesichert ist
    - Nachteil: Aktualisierung braucht mehr Zeit und ggf. Leistungsproblem wegen zu langer Wartzeiten

Konsistenzprotokolle

Urbildbasierte Protokolle (Primary-Based Protocols)

- Protokolle für entfernte Schreibvorgänge



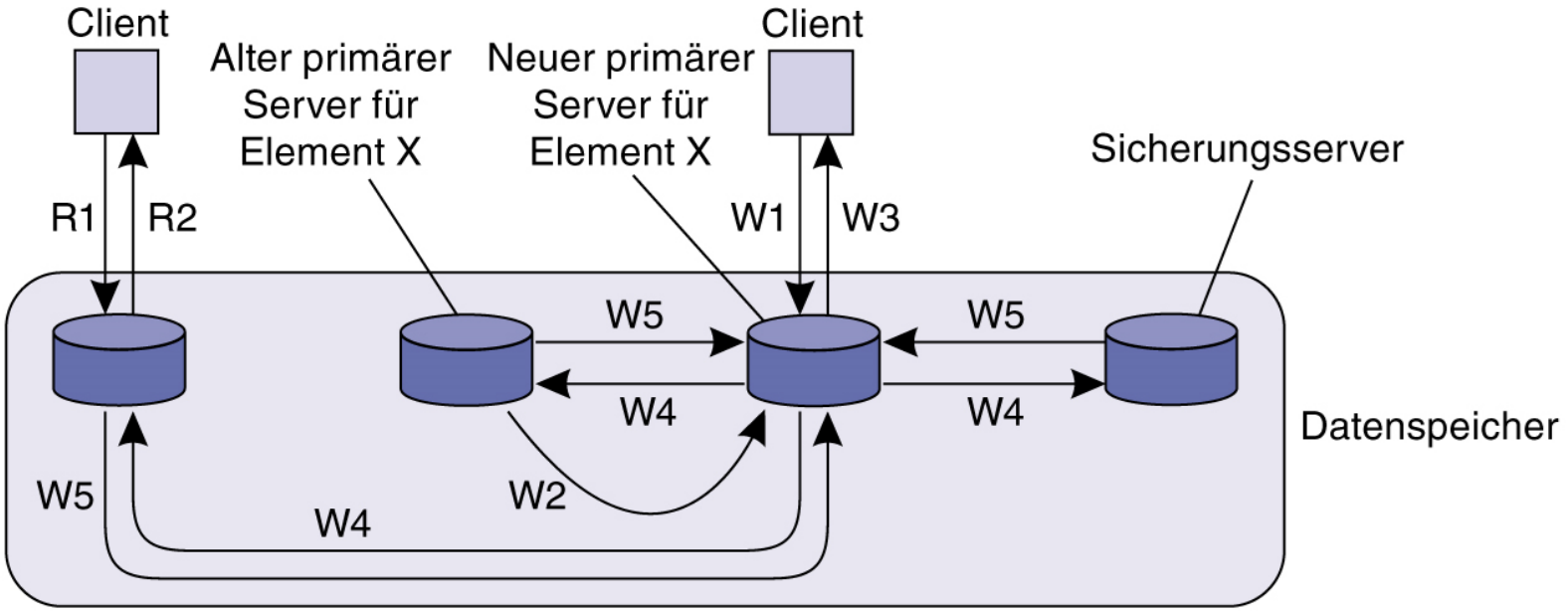
W1 – Schreibanforderung
W2 – Weiterleitung der Anforderung an den primären Server
W3 – Anweisung an die Sicherungen zur Aktualisierung
W4 – Aktualisierungsbestätigung
W5 – Bestätigung des abgeschlossenen Schreibvorgangs

R1 – Leseanforderung
R2 – Leserückmeldung

Konsistenzprotokolle

Urbildbasierte Protokolle (Primary-Based Protocols)

- Protokolle für lokale Schreibvorgänge



W1 – Schreibanforderung
W2 – Verschieben von Element X zum neuen primären Server
W3 – Bestätigung des abgeschlossenen Schreibvorgangs
W4 – Anweisung an die Sicherungen zur Aktualisierung
W5 – Aktualisierungsbestätigung

R1 – Leseanforderung
R2 – Leserückmeldung

Konsistenzprotokolle

Urbildbasierte Protokolle (Primary-Based Protocols)

- Protokolle für lokale Schreibvorgänge
  - Idee: primäre Kopie migriert zwischen Prozessen
  - Vorteil: mehrere Schreiboperationen leicht hintereinander ausführbar
  - Achtung: Leistungsgewinn nur bei Einsatz von nicht sperrenden Protokollen

Konsistenzprotokolle
Protokolle für replizierte Schreibvorgänge

- Idee: Schreiboperationen auf mehreren Replikaten möglich (Gegensatz zu Urbildbasierten Protokollen)
- Ansätze:
  - Aktive Replikation
  - Quorumgestützte Protokolle (auf mehrheitliche Abstimmung basierend)

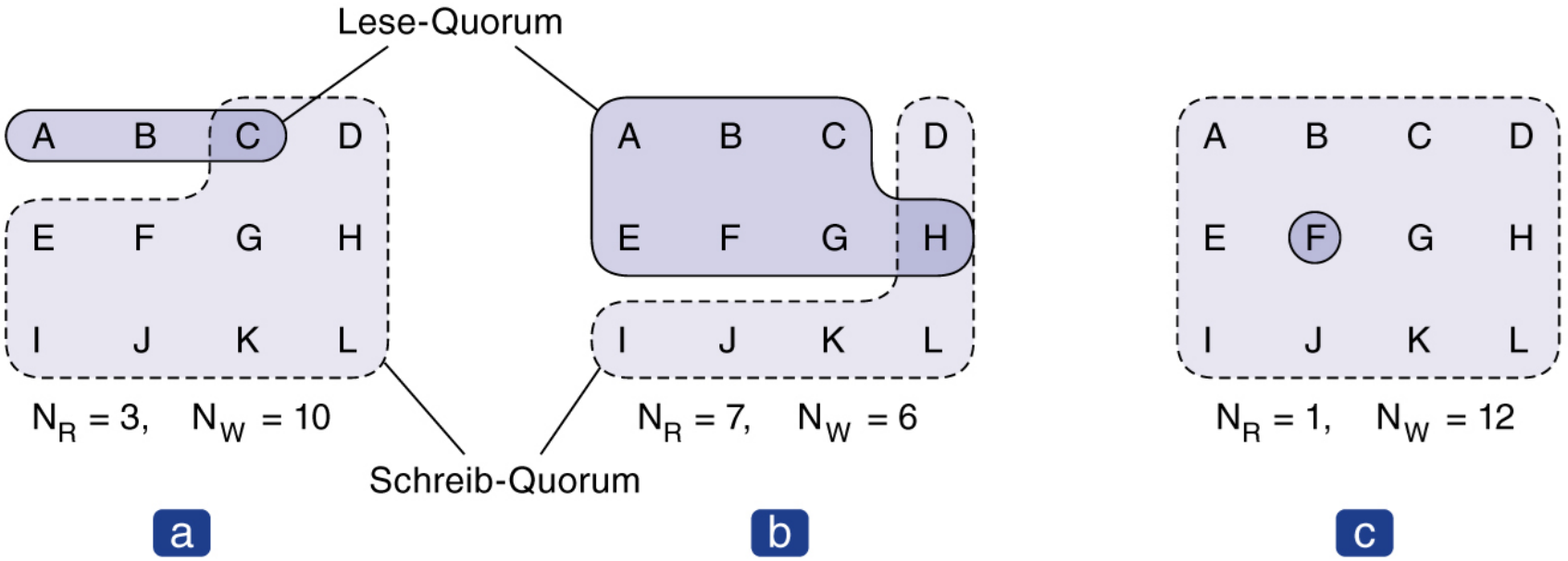
- Aktive Replikation
  - Aktualisierung in Form von Schreiboperationen an alle Replikate gesendet
  - Problem: Gleiche Reihenfolge ist nötig z.B. durch vollständig geordnetes Multicasting
    - Einsatz der logischen Uhr von Lamport oder
    - Einsatz eines Sequenzierers (zentraler Koordinator, der allen Operationen eindeutige fortlaufende Nummer gibt)
  - Problem nun: Skalierbarkeit

Konsistenzprotokolle
Protokolle für replizierte Schreibvorgänge

- Grundgedanke:
  - Clients bedürfen der Erlaubnis mehrerer Server, bevor sie ein repliziertes Datenelement lesen oder schreiben dürfen
- Einfache Realisierung: Es bedarf stets der Mehrheit von mindestens N/2+1
- Verallgemeinerung:
  - N – Anzahl aller beteiligter Server / aller Replikaten
  - N_R – Anzahl an nötigen Servern für ein Lesequorum
  - N_w – Anzahl an nötigen Servern für ein Schreibquorum
  - Bedingungen:
    1) $N_R + N_W > N$
    2) $N_W > N/2$
  - Bedingung 1): Verhinderung von Lese-Schreib-Konflikten
  - Bedingung 2): Verhinderung von Schreib-Schreib-Konflikten

Konsistenzprotokolle

Protokolle für replizierte Schreibvorgänge



Drei Beispiele für den Abstimmungsalgorithmus:
(a) eine korrekte Vorgabe für den Lese- und den Schreibvorgangssatz
(b) eine Wahl, die zu Schreib-Schreib-Konflikten führen kann
(c) eine korrekte Wahl, die als ROWA (Read-One, Write-All) bekannt ist

Quellenhinweis

- Vorlesung „Distributed Systems“ von Maarten van Steen
  - Thema: Consistency & Replication
    https://www.youtube.com/watch?v=pdxGtahoqlY

Themenüberblick

- Konsistenz und Replikation
- **Fehlertoleranz**
  - **Grundbegriffe**
  - Prozess-Resilienz

Fehlertoleranz - Einstiegsfragen

1) Was verstehen Sie unter Fehlertoleranz?

2) Welche anderen Umgangsformen hinsichtlich Fehlern sind noch denkbar?

3) Was ist der Unterschied zwischen einer Störung (fault), einem Fehler (error) und einem Ausfall (failure)?

4) Welche Arten von Ausfällen kann man unterscheiden?

5) Welche Grundansätze hinsichtlich der Fehlertoleranz in verteilten Systemen sind denkbar?

Fehlertoleranz - Grundbegriffe

- Fehlertoleranz
  - Tolerieren von Ausfallfehlern
  - Starke Verwandtschaft mit Verlässlichkeit
- Anforderungen an ein verlässliches System
  - Verfügbarkeit (Availablilty)
    - Wahrscheinlichkeit, dass das System zu einem bestimmten Zeitpunkt korrekt arbeitet (vgl. hochverfügbare Systeme)

Fehlertoleranz - Grundbegriffe

- Zuverlässigkeit (Reliability)
  - Aussage über fortlaufende Ausfallfreiheit bezogen auf Zeitintervall
  - z.B. Ausfall jede Stunde für 1ms (hochverfügbar, aber unzuverlässig)
  - z.B. keine Abstürze, aber 1 Monat Wartung (nicht hochverfügbar, aber zuverlässig)
- Funktionssicherheit (Safety)
  - Aussage über Auswirkungen im Fehlerfall (Fehler sollen nicht zur Katastrophe führen.)
- Wartbarkeit (Maintainability)
  - Aussage, wie leicht ein ausgefallenes System repariert werden kann

Fehlertoleranz - Grundbegriffe

- Wartbarkeit (Maintainability)
  - Aussage, wie leicht ein ausgefallenes System repariert werden kann
  - Aussage, über Aufwand bei Änderungen / Aktualisierungen am System

Fehlertoleranz - Grundbegriffe

- Umgangsformen hinsichtlich Fehlern
  - Fehlervermeidung (fault prevention)
    - Verhinderung des Auftretens von Fehlern
    - z.B. Schulung der Programmierer, Testen, etc.
  - Fehlertoleranz (fault tolerance)
    - Komponenten entwerfen, um das Auftreten von Fehlern zu verbergen
  - Fehlerbehebung (fault removal)
    - Reduzierung der Fehler hinsichtlich der Existenz, der Anzahl, des Schweregrades
  - Fehlervorhersage (fault forecasting)
    - Abschätzung der aktuellen Existenz, zukünftiger Vorfälle und hinsichtlich der Fehlerkonsequenzen
    - z.B. bei Kenntnis von Fehlerquellen → Abschätzung hinsichtlich des wirtschaftlichen Schadens beim Auftreten

Fehlertoleranz - Grundbegriffe

- Systemausfall (Failure)
  - Zusagen können nicht eingehalten werden
- Fehler (Error)
  - Teil des Systemzustandes, der zum Ausfall führen kann
- Störung (Fault)
  - Ursache eines Ausfalls
  - Herangehensweise an Störungen
    - Verhindern, Beheben, Vorhersagen von Fehlern
  - Fehlertoleranz: Trotz vorliegen bestimmter Störungen kann ein System seine Dienste bereitstellen.

Fehlertoleranz - Grundbegriffe

- Störung (Fault)
  - Unterscheidung:
    - Vorübergehende Störungen (Transient Faults) (z.B. Vogel durch Strahl eines Mikrowellensender)
    - Wiederkehrende Störungen (z.B. Wackelkontakt, …)
    - (siehe https://www.pcwelt.de/news/Wegen-altem-TV-Internet-Ausfaelle-im-ganzen-Ort-10887532.html)
    - Permanente Störungen (z.B. Soft- oder Hardwarefehler)

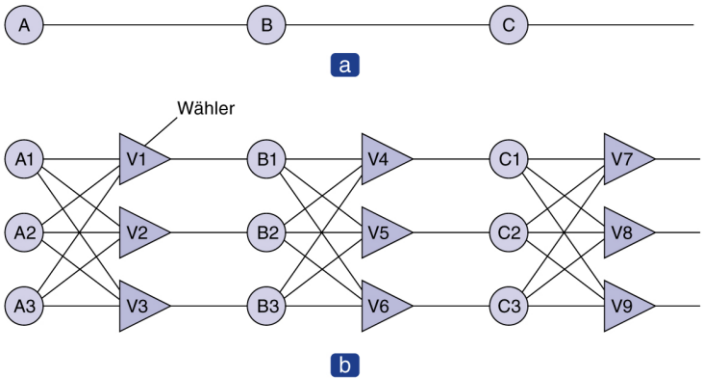
Fehlertoleranz - Fehlermodelle

Ausfallart	Beschreibung
Absturzausfall (Crash Failure)	Ein Server steht, hat aber bis dahin richtig gearbeitet. Der angebotene Dienst bleibt beständig aus (ständiger Dienstausfall).
Dienstausfall (Omission Failure) *Empfangsauslassung* *Sendeauslassung*	Ein Server antwortet nicht auf eingehende Anforderungen. Ein Server erhält keine eingehenden Anforderungen. Ein Server sendet keine Nachrichten.
Zeitbedingter Ausfall (Timing Failure)	Die Antwortzeit eines Servers liegt außerhalb des festgelegten Zeitintervalls.
Ausfall korrekter Antwort (Response Failure) Ausfall durch *Wertfehler (Value Failure)* Ausfall durch *Zustandsübergangsfehler (State Transition Failure)*	Die Antwort eines Servers ist falsch. Dieser Ausfall wird oft auch kurz Antwortfehler genannt. Der Wert der Antwort ist falsch. Der Server weicht vom richtigen Programmablauf ab.
Byzantinischer oder zufälliger Ausfall (Arbitrary oder Byzantine Failure)	Ein Server erstellt zufällige Antworten zu zufälligen Zeiten.

Fail-stop Failure (Ausfall-Stopp) – Dienst hört einfach auf
Fail-silent Failure (Ausfall durch Verschweigen) – andere Prozesse vermuten Absturz
...

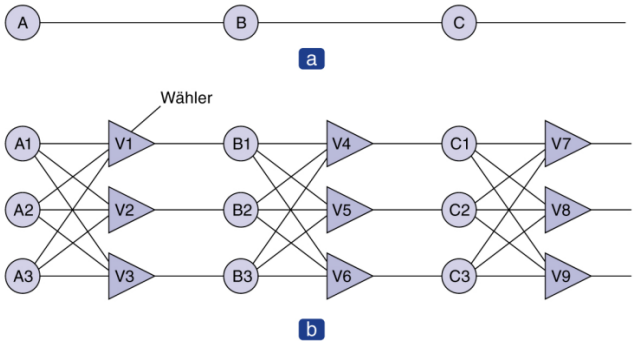
Fehlertoleranz

- Maskierung des Ausfalls durch Redundanz
  - Informationsredundanz
    - Zusätzliche Bits zwecks Wiederherstellung
    - z.B. Hamming-Code (siehe https://www.youtube.com/watch?v=X8jsijhllIA)
  - Zeitliche Redundanz
    - Aktion wird ggf. wiederholt
    - z.B. bei vorübergehende oder wiederkehrende Störungen



Fehlertoleranz

- Maskierung des Ausfalls durch Redundanz
  - Technische Redundanz
    - Zusätzliche Ausrüstung oder Prozesse zwecks Kompensation von ausgefallener oder fehlerhafter Komponenten
    - Realisierung via Hardware oder Software
    - Beispiel: Dreifache modulare Redundanz

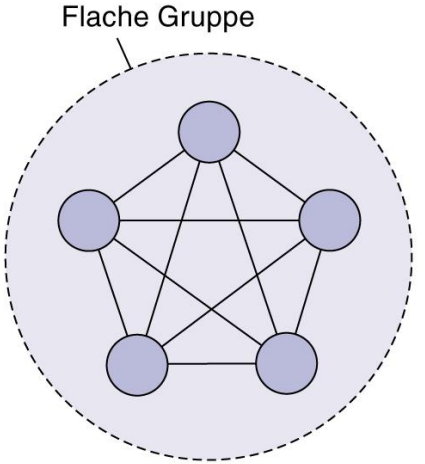


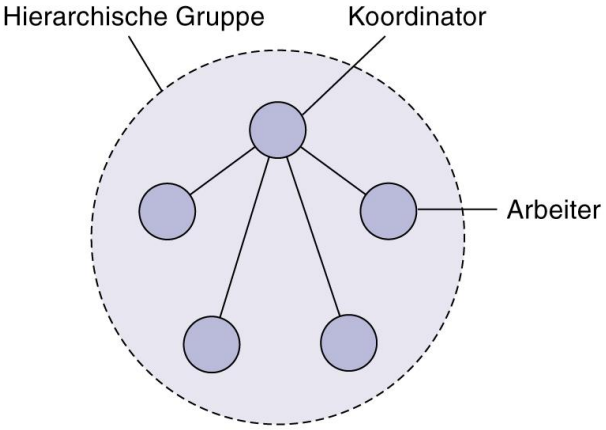
Themenüberblick

- Konsistenz und Replikation
  - Konsistenzprotokolle
- **Fehlertoleranz**
  - Grundbegriffe
  - **Prozess-Resilienz**

Prozess-Resilienz

- Lösungsansatz: Replizierung von Prozessen
  - Gruppierung von Prozessen (fehlertolerante Gruppe)
- Lineare Gruppe
  - Kein Chef und Entscheidungen stets gemeinsam
  - Vorteil: keinen einzelnen Ausfallpunkt
  - Nachteil: Entscheidungsfindung braucht Zeit
- Hierarchische Gruppe
  - Existenz eines Koordinators
  - z.B. Anforderung wird an besten Arbeiter weitergeleitet
  - Vorteil: schneller als lineare Gruppe
  - Nachteil: einzelner Ausfallpunkt (Koordinator)





*Resilienz: psychische Widerstandskraft; Fähigkeit, schwierige Lebenssituationen ohne anhaltende Beeinträchtigung zu überstehen

Prozessgruppen
Gruppenverwaltung

- Gruppenserver
  - Verwaltung der Gruppen und der Mitgliedschaften
  - Vorteil: leicht zu implementieren
  - Nachteil: einzelner Ausfallpunkt
- Alternative: Verteilte Verwaltung
  - Bedingt Existenz (zuverlässigen) Multicastings

Prozessgruppen
Gruppenverwaltung

- Betrachtungsaspekte:
  - Eintritt und Austritt einzelner Prozesse
    - Austritt mit oder ohne Ankündigung
    - Senden und Empfangen von Nachrichten synchron zum Ein- und Austritt
  - Kritische Größe einer Gruppe (Absturz mehrere Computer, so dass Gruppe nicht mehr funktioniert)

Prozessgruppen Designfragen

- Replikation
  - Urbildbasierte Protokolle
    - Hierarchische Strukturierung, Urbild koordiniert Schreibvorgänge
    - Absturz des Urbildes: Wahl unter den Backups
  - Protokolle für replizierte Schreibvorgänge oder quorumbasierte Schreibvorgänge
    - Anwendung in linearen Gruppen

Prozessgruppen Designfragen

- Anzahl an Replikationen
  - Bezeichnung: k-fehlertolerant
    - Kein Ausfall trotz Fehler in k Komponenten
  - k+1 Replikationen für k-Fehlertoleranz
    - Bei Absturz-, Dienst- und zeitlichem Ausfall
  - Mindestens 2k+1 Replikationen für k-Fehlertoleranz
    - Bei byzantinischen Ausfall oder Ausfall korrekter Antworten
- Bedingung: Anforderungen auf allen Servern in derselben Reihenfolge
  - Realisierung durch atomares Multicasting

Interludium
Problem der byzantinischen Generäle

- Referenz auf byzantinisches Reich (330-1453)
  - „… einem Ort (Balkan und die heutige Türkei), wo endlose Verschwörungen, Intrigen und Lügen in Herrscherkreisen als üblich galten.“
    (Tanenbaum und van Steen, Verteilte Systeme, 2. Auflage, S. 359)

(Byzantische Fehlertoleranz in Blockchains: https://www.youtube.com/watch?v=VWG9xcwjxUg)

Interludium
Problem der byzantinischen Generäle

- Problemstellung
  - Mehrere Divisionen geografisch verstreut (mit je einem General) belagern feindliches Lager
  - Übereinstimmung zwecks Angriff – Kommunikation zwischen Generälen via Boten
  - Übereinstimmung wichtig, da Angriff einiger Divisionen zur Niederlage führt
  - Verhinderung einer Übereinstimmung durch
    - Boten können vom Feind gefangen genommen werden (unzuverlässige Kommunikation) … aber in dem Fall ist das Problem ist nicht lösbar.
    - Unter den Generälen können Verräter sein (einstreuen irreführender Informationen)

(Byzantische Fehlertoleranz in Blockchains: https://www.youtube.com/watch?v=VWG9xcwjxUg)

Einigungsalgorithmen

- Beispiel für Relevanz von Einigung
  - Auswahl eines Koordinators
  - Entscheidung, ob eine Transaktion mit Commit festgeschrieben wird
  - Aufteilung von Aufgaben
  - ...
- Annahme: Prozess arbeiten nicht zusammen, um ein falsches Ergebnis zu produzieren.

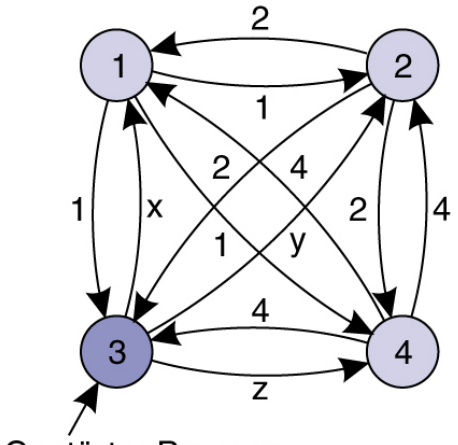
- Ziel verteilter Einigungsalgorithmen:
  - Alle nicht fehlerbehafteten Prozesse sind sich einig über einen Aspekt
  - Erreichen der Einigkeit in endlicher Anzahl von Schritten

Einigungsalgorithmen

- Lösung für das byzantinische Übereinstimmungsproblem
- Annahme: 4 Prozesse, darunter 1 fehlerhafter Prozess

- Schritt 1: Senden der Infos an alle anderen (a)
- Schritt 2: Sammeln aller erhaltenen Infos (b)
- Schritt 3: Ergebnisvektor an alle anderen (c)
- Schritt 4: Vergleich der erhaltenen Ergebnisvektoren

Ergebnis: (1,2,unbekannt,4)



a

1	Got(1, 2, x, 4)
2	Got(1, 2, y, 4)
3	Got(1, 2, 3, 4)
4	Got(1, 2, z, 4)

b

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

c

Erkennung von Ausfällen (Failure Detection)

- Ansatz 1:
  - Zustandsanfragen an andere Prozesse („Lebst Du noch?“)
  - Abwarten einer Rückantwort
  - Zeitüberschreitungen indiziert Absturz eines Prozesses
  - Problem: falsche Positivmeldungen möglich
- Ansatz 2:
  - Regelmäßiges Senden der Dienstverfügbarkeit an Nachbarn (Senden eines „Heartbeats“)

- Achtung: Unterscheidung zwischen Netzwerkausfällen und Knotenausfällen
  - Entscheidung über Ausfall nicht alleinig durch einzelnen Prozess

Erkennung von Ausfällen (Failure Detection)

- Szenario:
  - Rechner C erhält keine Nachricht eines anderen Rechners C*
  - Problem: Ist C* ausgefallen?
  - Aussage hinsichtlich: Absturzausfall, Dienstausfall oder zeitbedingter Ausfall

Erkennung von Ausfällen (Failure Detection)

- Unterscheidung
  - Asynchrone Systeme
    - Keine Aussage über Ausführungs- oder Auslieferungsgeschwindigkeiten
      → Detektion von Absturzausfällen nicht möglich
  - Synchrone Systeme
    - Ausführungs- oder Auslieferungsgeschwindigkeiten innerhalb vorgeschriebener Grenzen
      → Zuverlässige Detektion von Dienstausfall oder zeitbedingter Ausfall
  - Teilweise synchrone Systeme (partially synchronous systems)
    - Annahme über System: weitestgehend synchrones System auch wenn keine Zeitgrenzenvorgaben
      → Normalerweise - Detektion von Absturzausfällen möglich