NoSQL

Prof. Dr. Ingo Claßen

Hochschule für Technik und Wirtschaft Berlin

Warum NoSQL

- Unterstützung großer Datenmengen verwaltet durch Computer-Cluster
- Horizontale Skalierbarkeit
- Flexibler Umgang mit Schemata
- Bessere Anpassung an Datenstrukturen in Anwendungen
- Polyglot Persistence

Stärken relationaler Datenbanksysteme

- Ausgereift und bereits lange verfügbar
- Hoher Grad an Standardisierung
- Einfaches Datenmodell (Tabellen), gut geeignet für betriebswirtschaftliche Daten
- Deklarative Abfragesprache mit komplexen Operationen (SQL)
- Starke Unterstützung für Konsistenzsicherung
- Starke Unterstützung für Transaktionen (ACID)
- Starke Unterstützung für Datensicherheit

Schwächen relationaler Datenbanksysteme

- Einfaches Datemodell: nur Tabellen (*Impedance Mismatch*)
- Normalisierung erforderlich
- Schemaänderungen aufwendig
- Keine Unterscheidung von Objekten und Beziehungen
- Schwache Unterstützung für Rekursion
- Eingeschränkte Datentypen
- Unterstützung verteilter Transaktionen aufwendig (2PC)
- Schlechte horizontale Skalierbarkeit

Kategorisierung von NoSQL-Systemen

- Schlüssel-Wert-Systeme
  - Einfaches Datenmodell
  - Schlüssel üblicherweise Zeichenketten
  - Werte z.B. Listen, Mengen, Hashes, Blobs
- Spaltenfamilien-Systeme
  - Mischung aus Schlüssel-Wert-Systemen und spaltenorientierten Datenbanken
  - Mehrdimensionales assoziatives Array
- Dokumentendatenbanken
  - Üblicherweise JSON-Dateien als Dokumente
  - Geschachtelte Strukturen
- Graphdatenbanken
  - Knoten und Kanten als Datenelemente
  - Einfache Navigation im Graphen

Aspekte verteilter NoSQL-Systeme

- Skalierung
  - Horizontal, shared nothing
  - Nahtlos: Dynamisches Hinzufügen/Entfernen von Knoten
- Knoten-Hardware
  - Standard-Rechner (commoditiy hardware)
  - Heterogen
- Management
  - Dezentrale Kontrolle
  - Uniforme Einsetzbarkeit von Knoten
  - Last-Balanzierung

Verteilungstransparenz

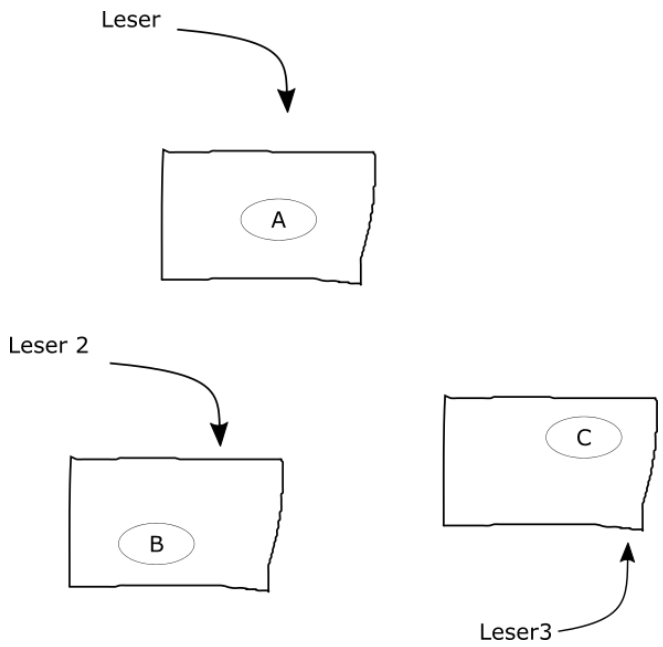
- Zugriffstransparenz
- Ortstransparenz
- Replikationstransparenz
- Framentierungstransparenz
- Migrationstransparenz
- Nebenläufigkeitstransparenz
- Fehlertransparenz

Fehler in verteilten Systemen

- Server-Fehler
- Fehler in der Nachrichtenübertragung
- Verbindungsfehler
- Netzwerkpartitionierung

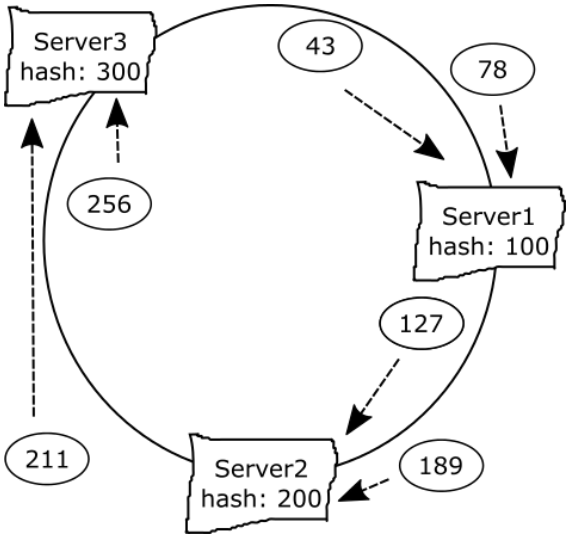
Sharding

- Daten auf verschiedenen Knoten
- Leser verteilen Arbeit auf Knoten
- Gemeinsam genutzte Daten auf gleichen Knoten (z.B. Aggregate)
- Verteilung Daten (z.B. geographisch)
- Zusammen mit Replikation

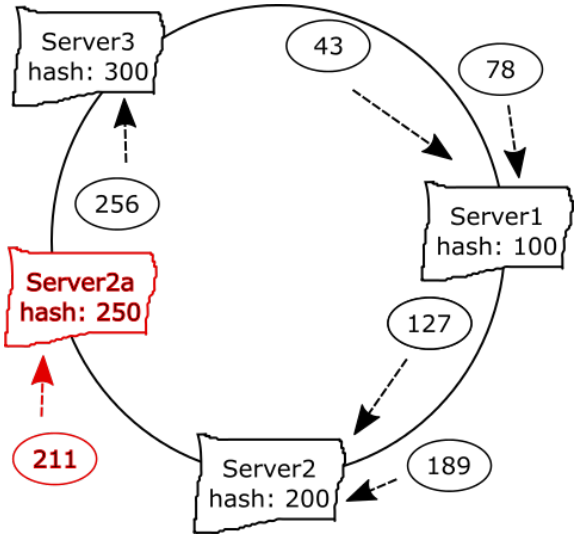


Allokation Objekte: Consistent Hashing

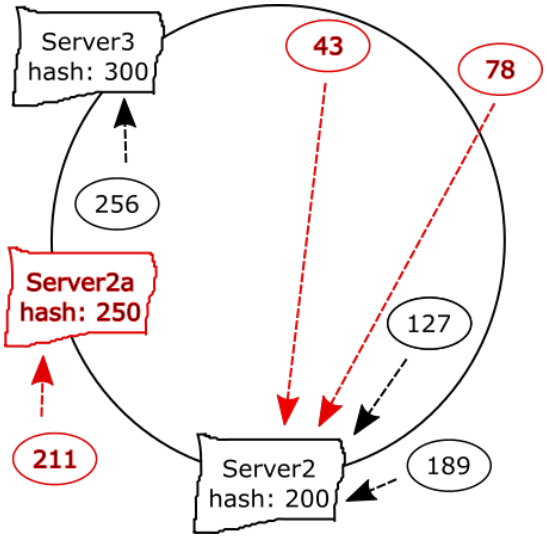
Zuordnung Objekte



Einfügen Knoten

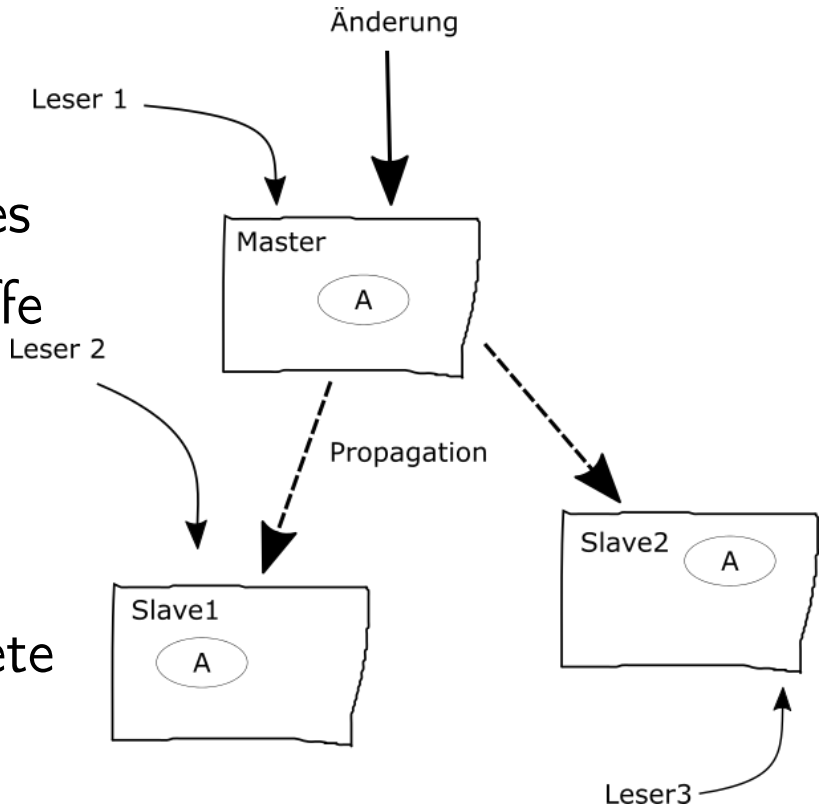


Löschen Knoten



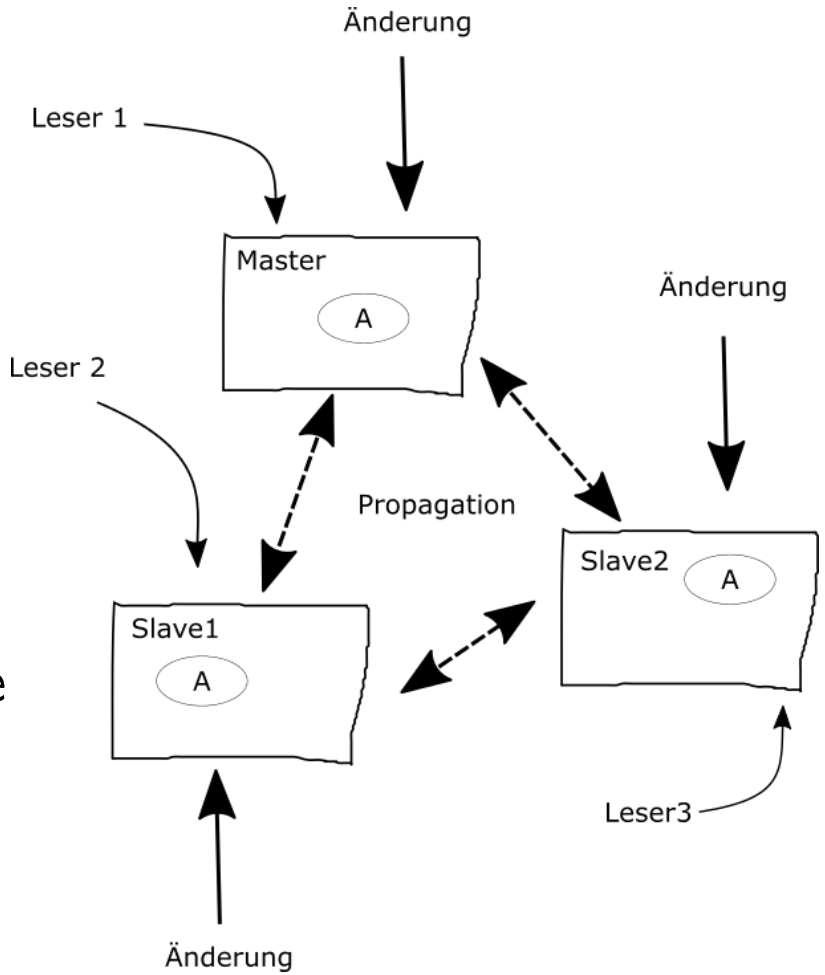
Master-Slave-Replikation

- Änderungen nur auf Master
- Propagation Änderungen auf Slaves
- Skalierung in Bezug auf Lesezugriffe
- Verfügbarkeit – Lesen bei Ausfall Master möglich
- Sicherungskopien der Daten
- Inkonsistenzen beim Lesen (veraltete Zustände)



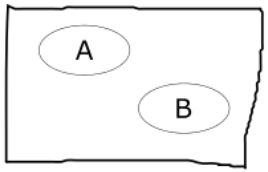
Peer-To-Peer-Replikation

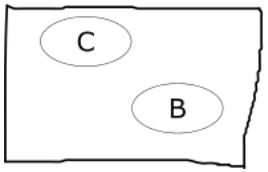
- Vor- und Nachteile wie Master-Slave-Replikation
- Weiterer Vorteil: Skalierung beim Schreiben
- Weiterer Nachteil: Schreibkonflikte

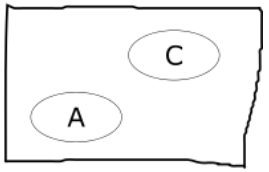


Replikation und Sharding

- Skalierung + Ausfalltolernanz
- Verschiedene Master für verschiedene Datenelemente möglich







CAP - Consistency, Availability, Partition Tolerance

- Konsistenz (Consistency)
  - Konsistenter Zustand im verteilten System
  - Änderung auf einem Knoten
  - Lesezugriff auf replizierten Knoten liefert geänderten Wert
- Verfügbarkeit (Availability)
  - System bietet akzeptable Reaktionszeit
  - Auch bei Ausfall von Knoten und Netzwerkverbindungen
- Ausfalltoleranz (Partition Tolerance)
  - Bei Ausfall von Knoten / Netzwerkverbindungen: kein Ausfall Gesamtsystem

Zusammenspiel von C, A und P

1. Kein Ausfall im System
   - C und A kann sichergestellt werden
   - Replikation der Daten kann erfolgen
2. Ausfall im System – Entscheidung zwischen C und A
   - Entscheidung für C
     - Operationen auf allen Knoten ablehnen
     - Keine Verfügbarkeit
   - Entscheidung für A
     - Operationen auf verfügbaren Knoten zulassen
     - Erzeugt inkonstistenten Zustand im Gesamsystem
     - Knoten können verschiedene Zustände für gleiches Objekt haben
     - Konstistenz später herstellen
     - (Eingeschränkte) Verfügbarkeit
   - Keine absolute Entscheidung zwischen C und A
   - Verschiedene Strategien möglich

Base - Basic Available, Soft State, Eventually Consistent

- Basic Available
  - (Eingeschränkte) Verfügbarkeit als oberstes Ziel
- Soft State
  - Fenster der Inkonsistenz
- Eventually Consistent
  - Irgendwann wird ein konsistenter Zustand erreicht

Konsistenz aus Nutzersicht

- Beteiligte
  - Verteiltes Speichersystem
  - Prozess A schreibt und liest
  - Prozesse B und C unabhängig von A, schreiben und lesen auch
- Konsistenzmodelle
  - Starke Konsistenz
    - Nach Änderung: A, B und C sehen immer den neuen Wert
    - Alle Replikate müssen aktualisiert sein
  - Schwache Konsistenz
    - A, B oder C sehen den alten Wert
    - z.B. beim Lesen eines Replikats
    - Eventual Consistency ist eine Form schwacher Konsistenz

Variationen von Eventual Consistency

- Causal consistency
  - B hängt kausal von A ab: B sieht neuen Wert
  - C sieht möglicherweise alten Wert
- Read-your-writes consistency
  - A hat Wert geändert: A sieht immer neuen Wert
- Session consistency
  - Innerhalb Sitzung: A hat ändert: A sieht neuen Wert
  - AußerhalbSitzung: A sieht möglicherweise alten Wert
- Monotonic read consistency
  - A hat neuen Wert gelesen: A sieht immer neuen Wert danach
- Monotonic write consistency
  - Serialisierung der Schreibzugriffe innerhalb eines Prozesses

Konsistenz aus Server-Sicht

- Definitionen
  - N = Anzahl Knoten
  - W = Anzahl Knoten Änderungsbestätigung (W-Quorum)
  - R = Anzahl Knoten Lesebestätigung (R-Quorum)
- Alternativen
  - W = N: Fokus auf Konsistenz
  - W = 1: Fokus auf Verfügbarkeit
  - W+R > N: Starke Konsistenz
  - W+R <= N: Schwache Konsistenz

Literatur

- **NoSQL - Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken**, Edlich, Friedland, Hampe, Brauer, Brückner, Hanser, 2011
- **Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement**, Redmont, O'Reilly UK Ltd, 2012
- **NoSQL Distilled**, Saldage, Fowler, Addison Wesley, 2012
- **Advanced Data Management**, Wiese, De Gruyter, 2015
- **Eventually Consistent**, Vogel, Acm Queue, 2008
- **CAP Twelve years later**, Brewer, IEEE, 2012