Skalierbarer und zuverlässiger Zugriff auf feingranulare Daten

Christopher Eibel

christopher.eibel@informatik.stud.uni-erlangen.de

Ausgewählte Kapitel der Systemsoftware: Cloud Computing (SoSe 2010)
Friedrich-Alexander-Universität Erlangen-Nürnberg

10. Juni 2010

Gliederung

Gliederung

Motivation (1)

Daten in der Cloud

- Clouds bestehen aus vielen weltweit vernetzten **Datenzentren**
- Datenzentren: Beherbergen viele miteinander vernetzte Rechner
- Verwendung günstiger **Commodity-Hardware**
- Bereitstellung von theoretisch **unbegrenzten Speicherkapazitäten**

Motivation (2)

Aber: Ausfallsicherheit? Zuverlässigkeit?

Alles fällt aus!

- **Rechner**, gerade bei günstiger Commodity-Hardware (Hardware-Probleme, z.B. Head-Crash)
- **Datenzentren** (Katastrophen, Stromausfälle, Kühlungsprobleme)
- **Netzwerkverbindungen** zwischen den Rechnern oder den Datenzentren selbst (einschließlich Router, Switches, etc.)

Quelle: [2]

Motivation (3)

Aber: Ausfallsicherheit? Zuverlässigkeit?

„Zeit ist Geld“

- Große Firmen wie Amazon oder eBay haben ein enormes Datenaufkommen und hohe Besucherzahlen
- Ausfälle und ein träges System versus Kundenzufriedenheit
- Zuverlässigkeit: Ein fehlerhaftes System darf keine Softwarefehler wie fehlerhafte Kreditkartenabbuchungen hervorrufen (Kundenvertrauen)
- Kundenzufriedenheit und -vertrauen wichtig! (Umsatzgedanke)

Spezielle Datenverwaltungssysteme nötig

- Müssen mit solchen Ausfällen und der Verteiltheit umgehen können
- **Hochverfügbarkeit** muss trotzdem ermöglicht werden
- Ansatz auf der Ebene von **Infrastructure-as-a-Service** (IaaS)

Motivation (4)

Warum ist ein relationales Datenbankverwaltungssystem (RDBVS/RDBMS) ungeeignet?

Umfangreiches Funktionsangebot

- ACID-Eigenschaften
  - Atomarität
  - Konsistenzerhaltung
  - Isolation
  - Dauerhaftigkeit
- Kann sehr große Datenmengen verwalten
- Komplexe Anfragen möglich
- Mehrbenutzerfähigkeit
- Hohes Maß an Datensicherheit

u.v.m.

Motivation (4)

Warum ist ein relationales Datenbankverwaltungssystem (RDBVS/RDBMS) ungeeignet?

Umfangreiches Funktionsangebot

- ACID-Eigenschaften
  - Atomarität
  - Konsistenzerhaltung
  - Isolation
  - Dauerhaftigkeit
- Kann sehr große Datenmengen verwalten
- Komplexe Anfragen möglich
- Mehrbenutzerfähigkeit
- Hohes Maß an Datensicherheit

  u.v.m.

Problem

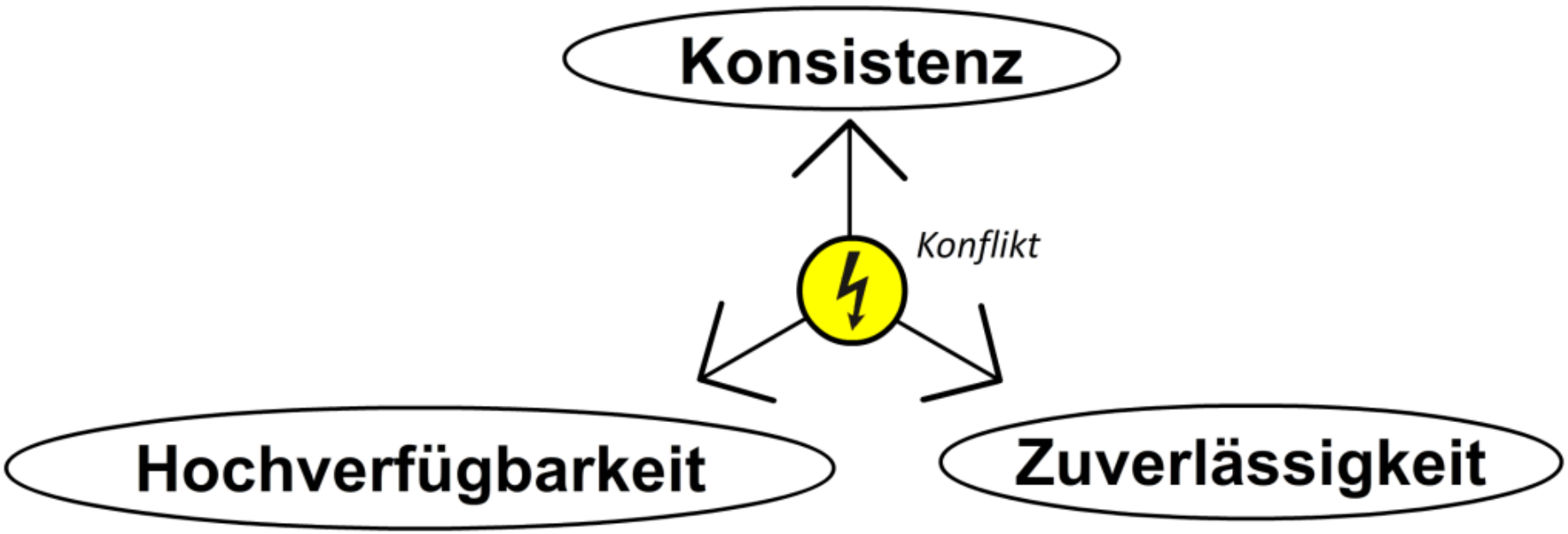
Hochverfügbarkeit? Skalierbarkeit? → Wird zum Flaschenhals

Konsistenz (1)

Konflikt

Hochverfügbarkeit, Zuverlässigkeit und Konsistenz

- Kriterien können nicht gleichzeitig erfüllt werden
- Mindestens ein Kriterium muss abgeschwächt werden

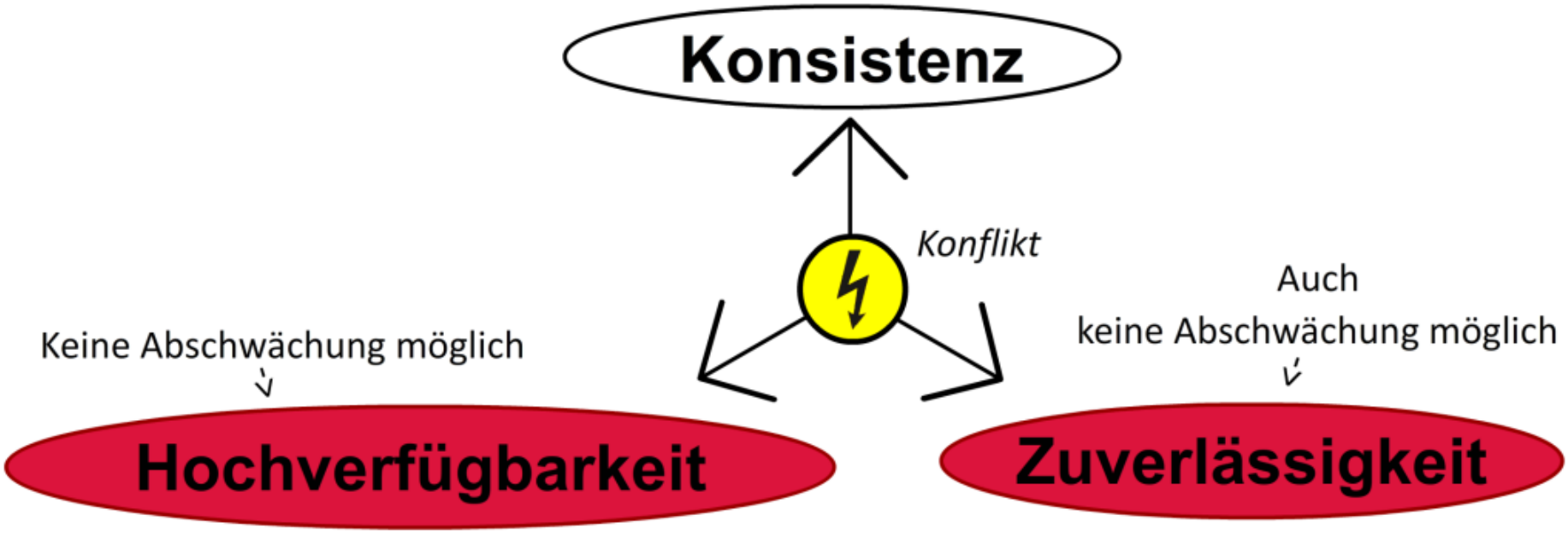


Konsistenz (1)

Konflikt

Hochverfügbarkeit, Zuverlässigkeit und Konsistenz

- Kriterien können nicht gleichzeitig erfüllt werden
- Mindestens ein Kriterium muss abgeschwächt werden

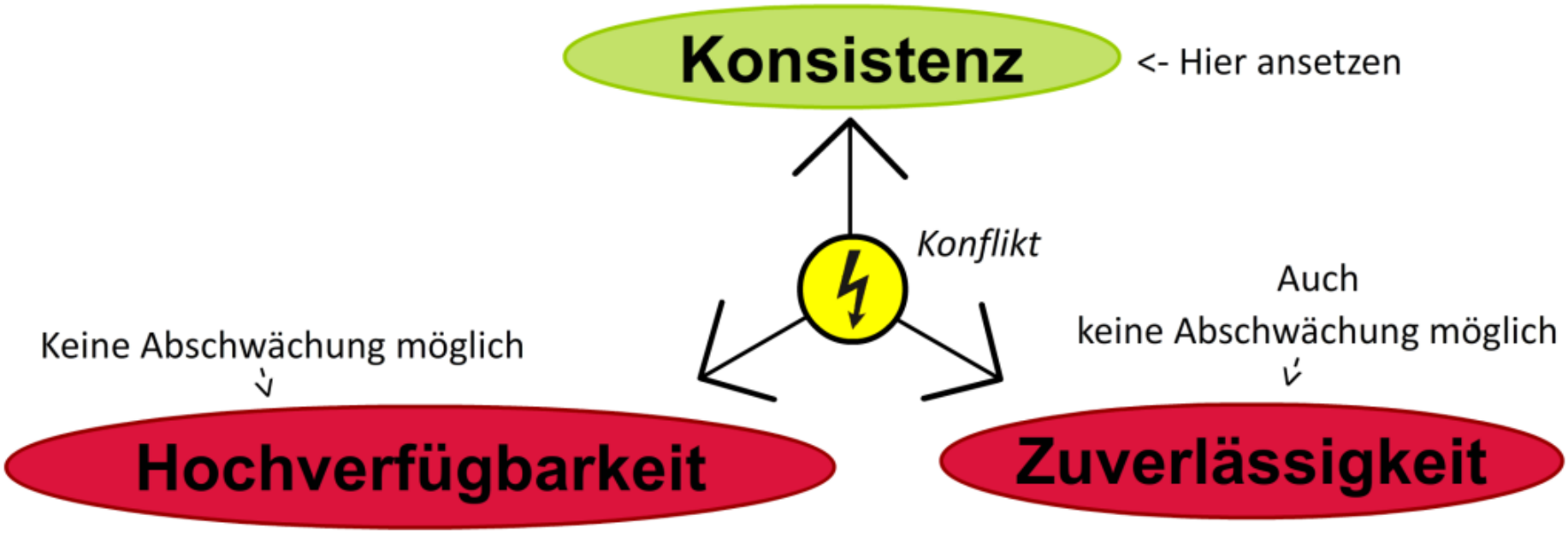


Konsistenz (1)

Konflikt

Hochverfügbarkeit, Zuverlässigkeit und Konsistenz

- Kriterien können nicht gleichzeitig erfüllt werden
- Mindestens ein Kriterium muss abgeschwächt werden



Konsistenz (2)

Konsistenzarten

Strenge Konsistenz

Nach erfolgreichem Schreibvorgang haben alle Replikate die gleiche Sicht auf die Daten

Konsistenz (2)

Konsistenzarten

Strenge Konsistenz

Nach erfolgreichem Schreibvorgang haben alle Replikate die gleiche Sicht auf die Daten

Schwache Konsistenz

Inkonsistente Zustände zulässig; Reihenfolgen der Änderungen nicht spezifiziert

Konsistenz (2)

Konsistenzarten

Strenge Konsistenz

Nach erfolgreichem Schreibvorgang haben alle Replikate die gleiche Sicht auf die Daten

Schwache Konsistenz

Inkonsistente Zustände zulässig; Reihenfolgen der Änderungen nicht spezifiziert

Letztendliche Konsistenz (Eventual Consistency)

- Liegt zwischen strenger und schwacher Konsistenz
- $\exists$ ein Zeitfenster mit inkonsistenten Zuständen
- Irgendwann werden diese inkonsistenen Zustände aufgelöst
- Guter Kompromiss für ein hochverfügbares Dateisystem

Gliederung

Amazon Dynamo

„Dynamo: Amazon's **Highly Available Key-value Store**“ [1]

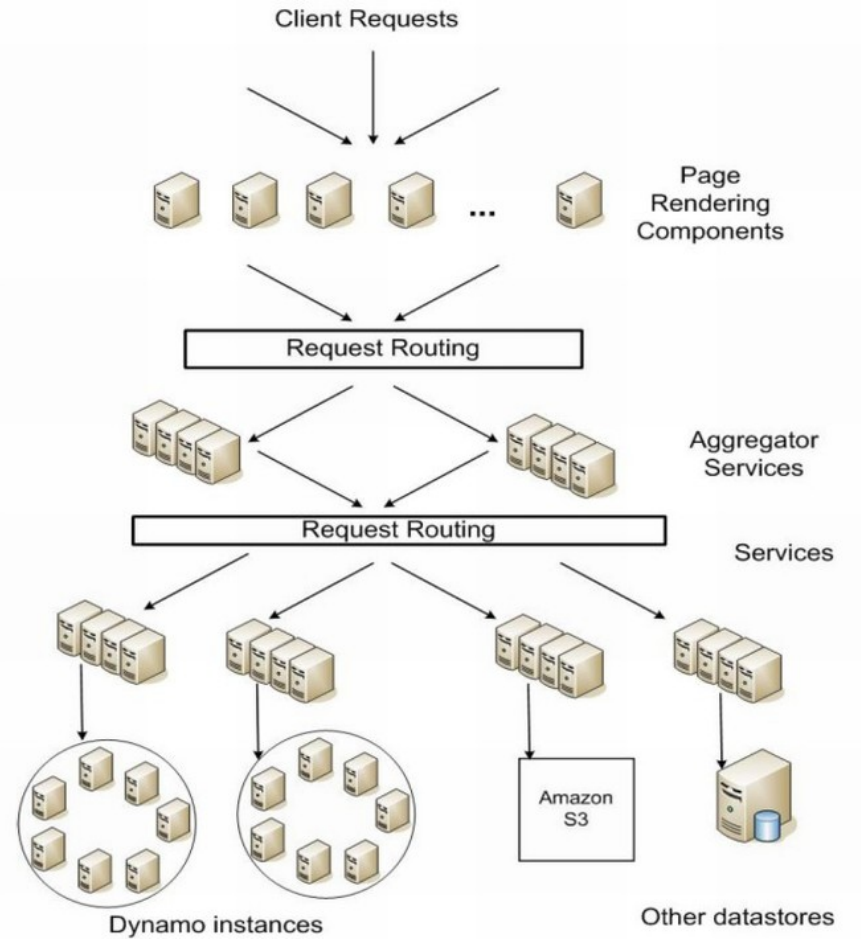
- **Hochverfügbarkeit** soll gewährleistet werden
- Speicherung mittels **Schlüssel-Werte-Paaren**
- **Feingranulare** Daten ($\leq$ 1 MB)

- **Verteiltes** Dateisystem
- Ausnutzung der **letztendlichen Konsistenz**
- Wird nur von Amazon-internen Diensten genutzt (sozusagen in einer Private Cloud)
- Anwendungen: Produktkatalog, Warenkorb (Shopping-Cart-Service), Produktinformationen, Bestseller-Listen

Amazons Architektur (1) – Client-Anfrage

Ablauf einer Client-Anfrage

1. Seiten-Erstellungs-Komponenten (Page-Rendering-Components) nehmen Anfrage entgegen
2. Seiten-Erstellungs-Komponenten stellen selbst Anfragen an die Sammel-Dienste (Aggregator-Services)
3. Aggregator-Services sammeln Daten, die auf den Datenbanken verteilt sind

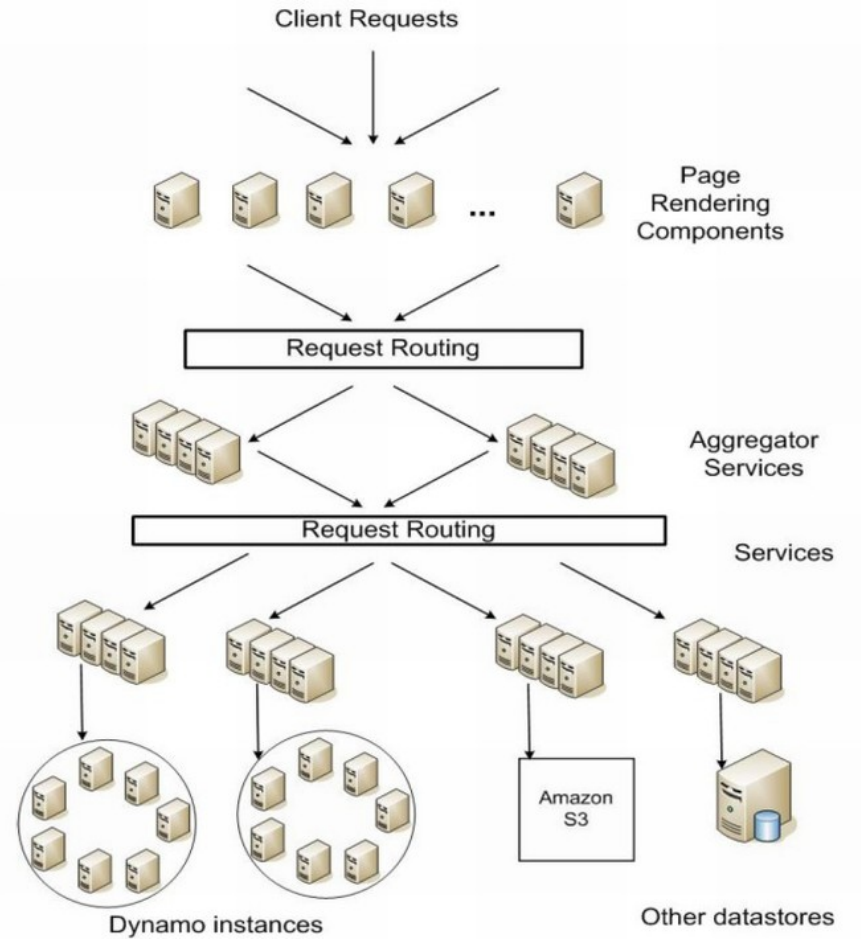


Quelle: [1]

Amazons Architektur (2) – Client-Anfrage (Fortsetzung)

Ablauf einer Client-Anfrage

4. Hochreichen der Daten an die Seiten-Erstellungs-Komponenten
5. Erstellung der Webseite
6. Auslieferung an den Client
7. Seite wird im Browser angezeigt

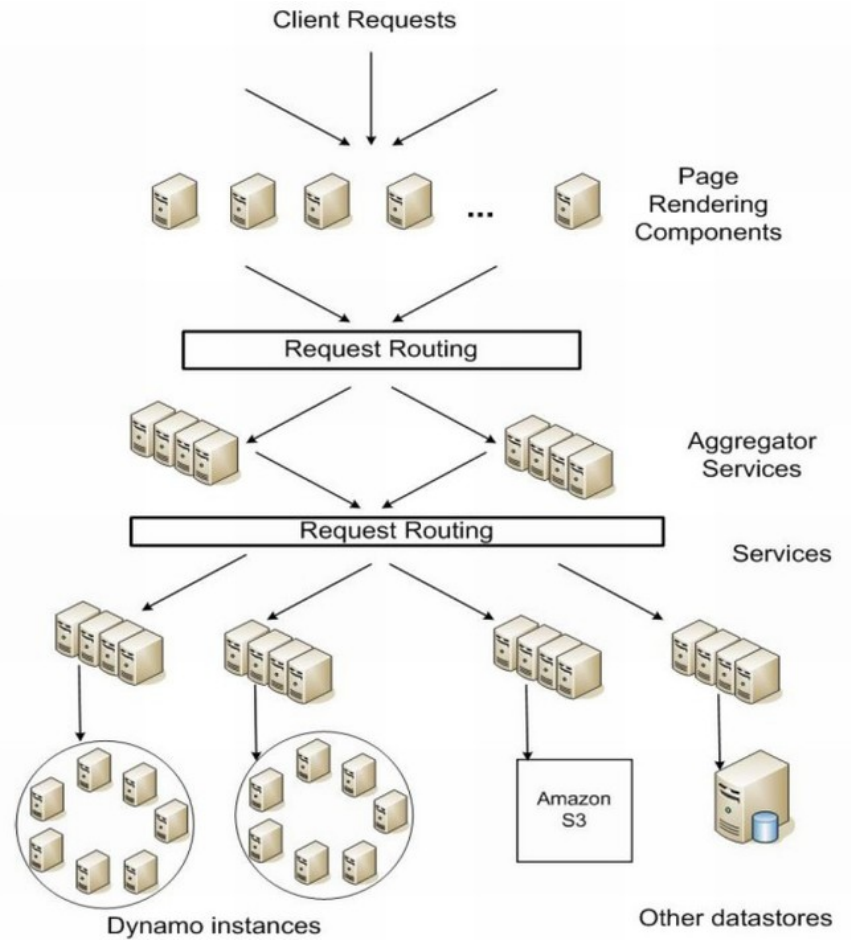


Quelle: [1]

Amazons Architektur (3) – SLAs

Service Level Agreements (SLAs)

- Auch: Dienstgütevereinbarung (DGV); Verträge zwischen den einzelnen Ebenen
- Entscheidet über Funktionalität, Geschwindigkeit und Kosteneffizienz
- Strengere SLAs mit zunehmender Tiefe
- Statt Median oder Durchschnitt: 99,9% („three nines“) – möglichst **alle** Anfragen innerhalb einer bestimmten Zeitspanne bedienen.



Quelle: [1]

Vereinfachungen

Vereinfachende Annahmen

- Keine Public Cloud, d.h. komplette Infrastruktur vertrauenswürdig: Keine Mechanismen zur Authentifizierung und Autorisierung nötig
- Daten werden ohne spezielles Schema abgespeichert
- Nur einfache Operationen (`put()` und `get()`)
- Keine relationale Anfragesprache nötig

Schnittstelle – Wie sehen diese „einfachen Operationen“ aus?

- Lesen: `get(byte[] key)`
- Schreiben: `put(byte[] key, Context c, byte[] object)`
- Byte-Arrays: Daten werden ohne speziellen Typ abgespeichert.
- Kontext: Metadaten (z.B. Vektoruhren)

Anforderungen

Wichtige Kriterien

- **Schnelle Antwortzeiten**, selbst bei vielen Knoten (Rechnern)
- **Symmetrie**: Alle Knoten haben die gleichen Aufgaben
  $\Rightarrow$ Skalierbarkeit und vereinfachte Instandhaltung
- **Dezentralisierung**: Ausfallsicherheit erhöhen, Skalierbarkeit
- **Heterogenität**: Abweichungen in Hard- und Software

Gliederung

Partitionierung und Replikation (1)

Grundlegendes zur Partitionierung

- **Zielsetzung**: Lastausgleich und Skalierbarkeit
- **Abbildungsvorschrift**: Wert $\rightarrow$ Schlüssel $\rightarrow$ Hashwert
- Die Daten werden abhängig von ihrem Schlüsselwert auf Rechner **verteilt** und **repliziert**
- Hashfunktion bildet Werte auf einen Ring ab
  $\Rightarrow$ Rechner werden logisch auf diesen Ring verteilt
- **Konsistentes** Hashing

Partitionierung und Replikation (2)

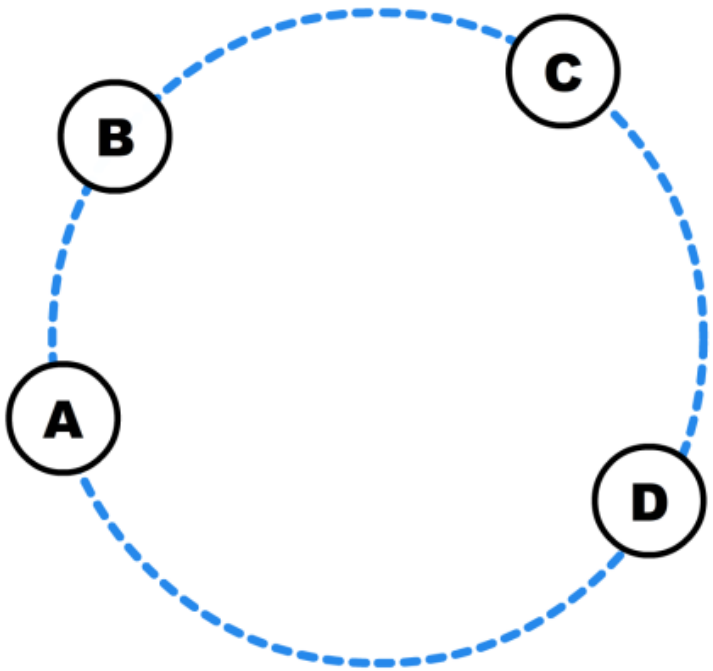
Zufällige Verteilung der Knoten (1)

Knoten- bzw. Rechnermenge $M = \{A, B, C, D\}$ auf den Ring verteilen.

Partitionierung und Replikation (2)

Zufällige Verteilung der Knoten (1)

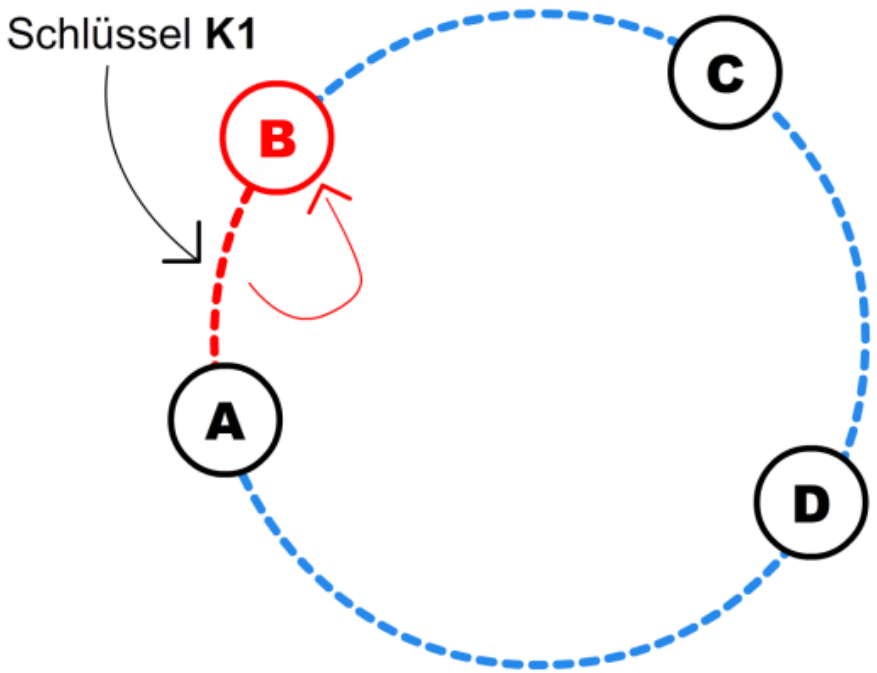
Knoten- bzw. Rechnermenge M = {A, B, C, D} auf den Ring verteilen.



Partitionierung und Replikation (3)

Zufällige Verteilung der Knoten (2)

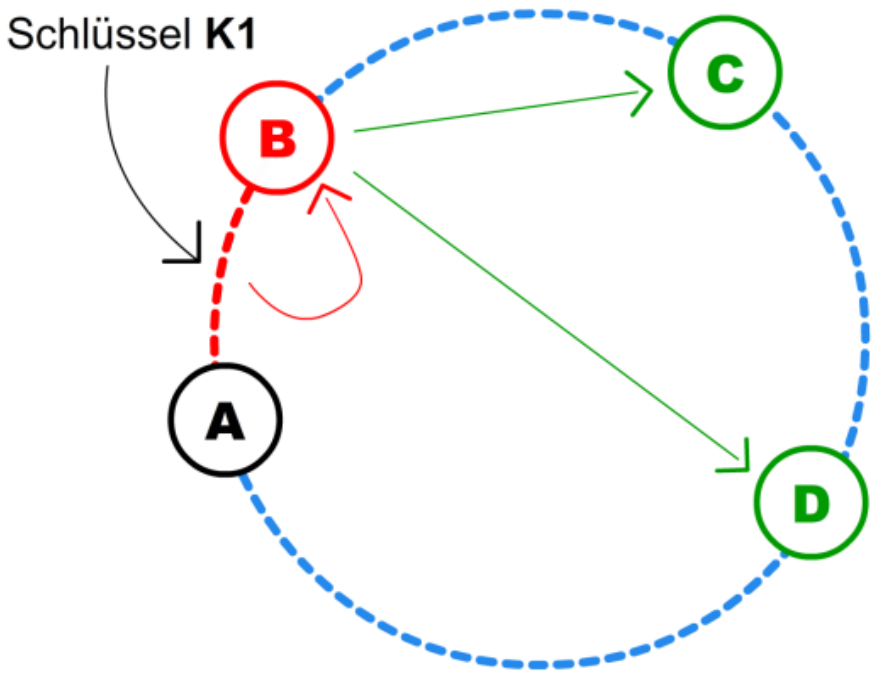
Ein Wert mit dem Schlüssel **K1** wird hinzugefügt.



Partitionierung und Replikation (4)

Zufällige Verteilung der Knoten (3)

Replikation mit **N = 3**, d.h. drei Replikate pro Datum. B ist Koordinator.



Partitionierung und Replikation (5)

Virtuelle Knoten (1)

Heterogenität

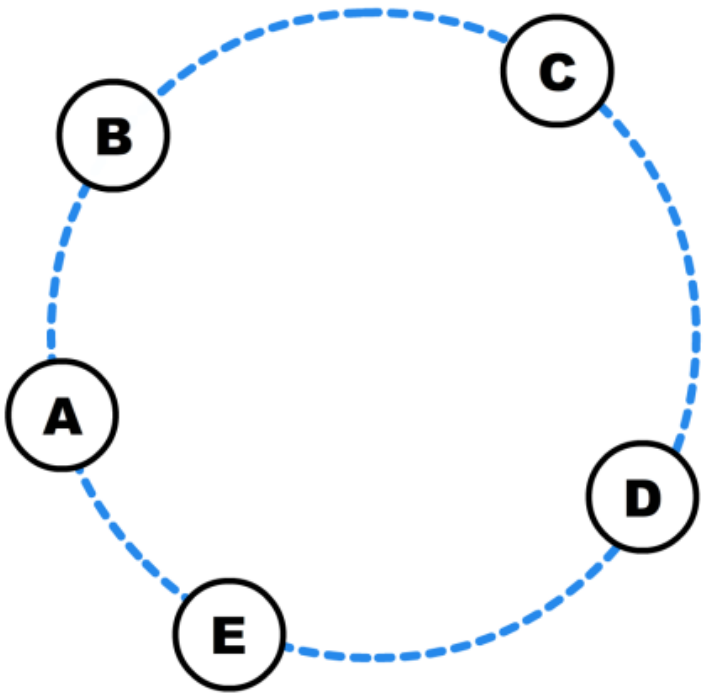
Rechner bieten unterschiedliche Speicherkapazitäten und Rechenleistungen.

→ Rechnern anhand mehrerer **virtueller Knoten** auf dem Ring einen passenden, gut verteilten Wertebereich zuweisen

Partitionierung und Replikation (6)

Virtuelle Knoten (2)

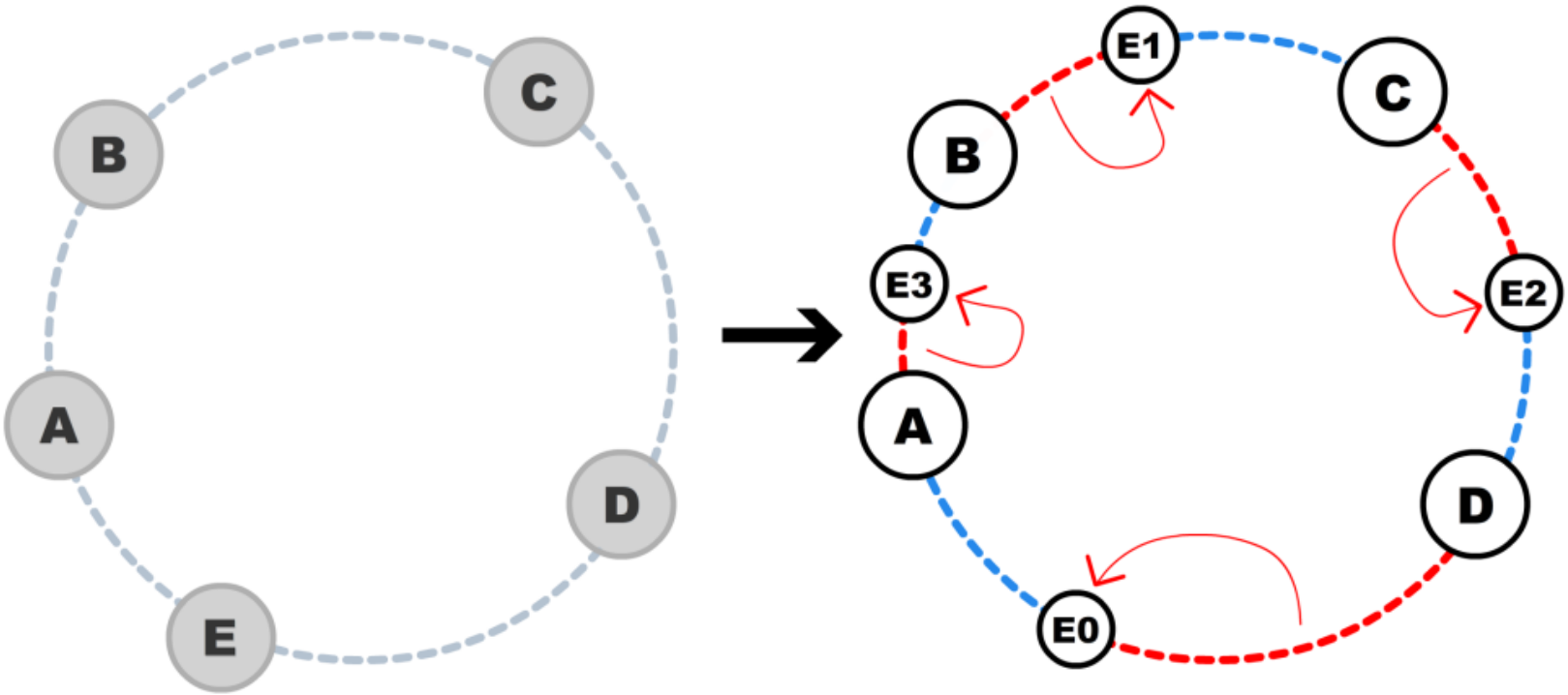
Neuer Knoten **E** wird aufgeteilt auf **E0** - **E3**.



Partitionierung und Replikation (6)

Virtuelle Knoten (2)

Neuer Knoten **E** wird aufgeteilt auf **E0** - **E3**.



Partitionierung und Replikation (7)

Preference-List (Vorzugsliste) (1)

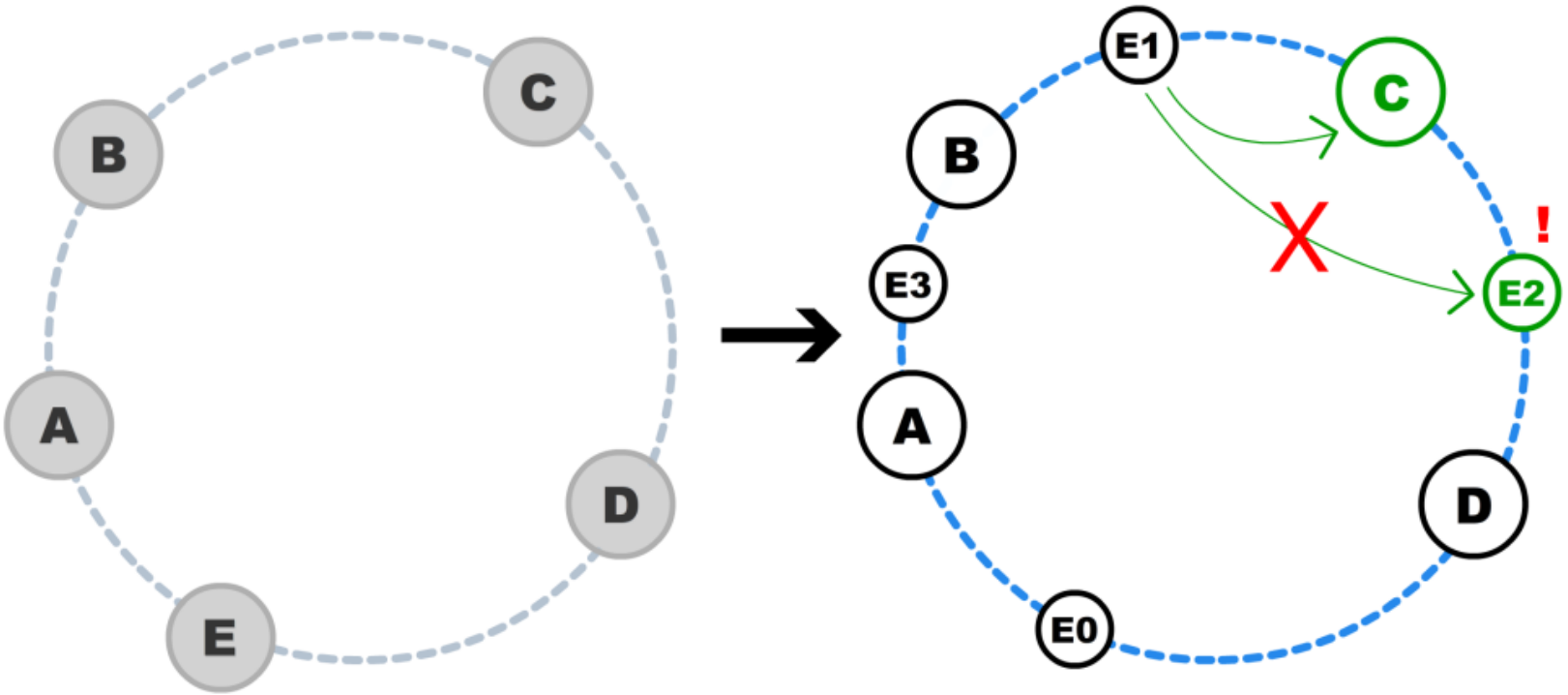
Vorzugsliste

- Liste von Knoten, auf die bevorzugt repliziert werden soll
- Enthält mehr als N Knoten, damit bei Ausfällen die gewünschte Anzahl an Replikaten trotzdem erreicht werden kann
- Auswahlentscheidung: Die virtuellen Knoten möglichst auf **unterschiedliche physikalische Rechner** in **unterschiedlichen Datenzentren** verteilen

Partitionierung und Replikation (8)

Preference-List (Vorzugsliste) (2)

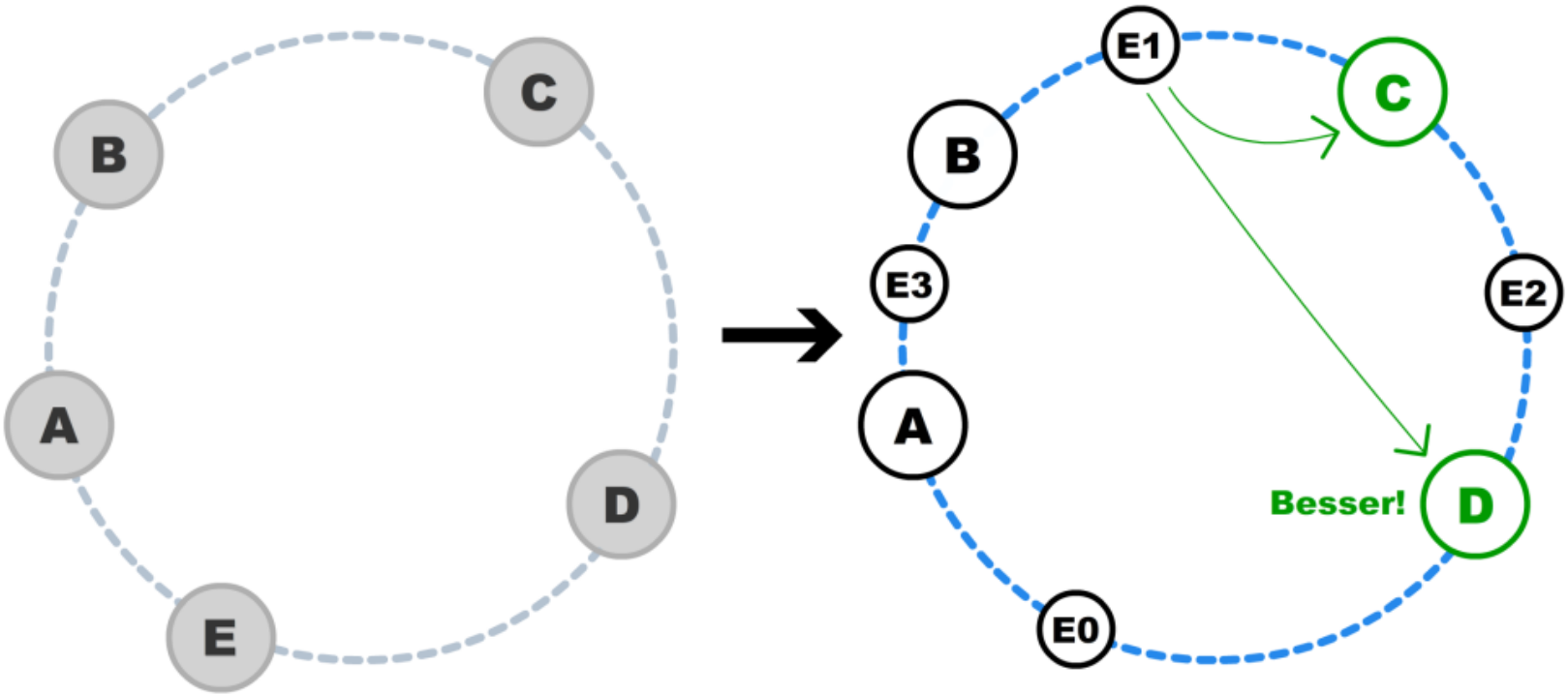
E1 repliziert nicht auf die direkten Nachfolger, sondern überspringt E2.



Partitionierung und Replikation (8)

Preference-List (Vorzugsliste) (2)

E1 repliziert nicht auf die direkten Nachfolger, sondern überspringt E2.



Partitionierung und Replikation (9)

Optimierte Partitionierungsstrategie (1)

Problem

- Hinzufügen eines Knotens führt zur weiteren Partitionierung des Rings
- Umfrangreiche Kopiervorgänge nötig
- Bei Lastspitzen müssen neue Knoten aber schnell hinzugefügt werden

Partitionierung und Replikation (9)

Optimierte Partitionierungsstrategie (1)

Problem

- Hinzufügen eines Knotens führt zur weiteren Partitionierung des Rings
- Umfrangreiche Kopiervorgänge nötig
- Bei Lastspitzen müssen neue Knoten aber schnell hinzugefügt werden

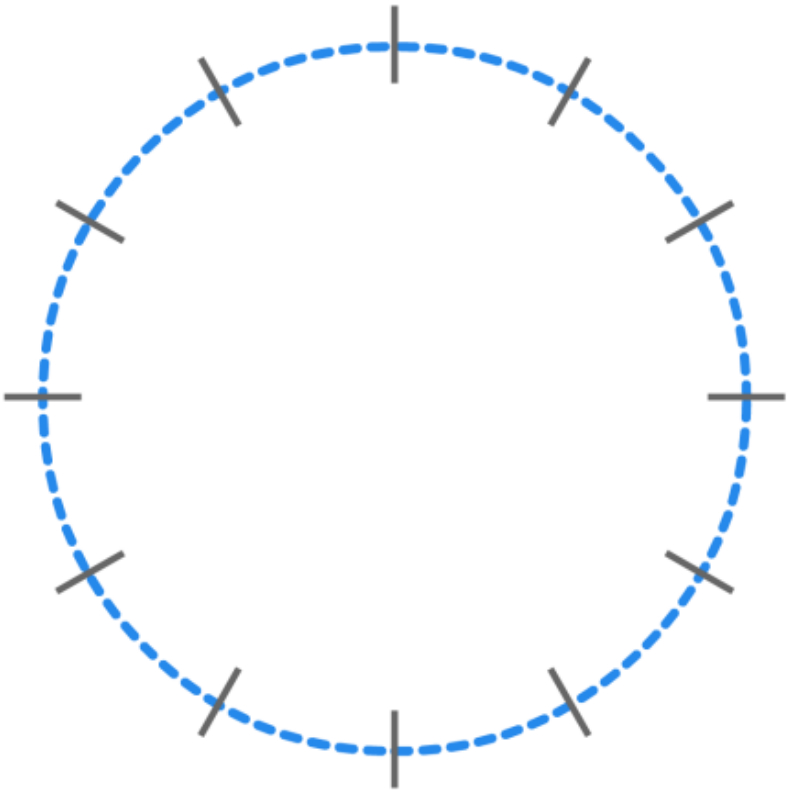
Lösung

- Entkopplung der Partitionierung und Datenplatzierung
- Partitionierungsbereich in Q feste Abschnitte einteilen ($Q \gg N$)
- Jeder Knoten erhält Q/S virtuelle Knoten (S = Anzahl der Knoten)

Partitionierung und Replikation (10)

Optimierte Partitionierungsstrategie (2)

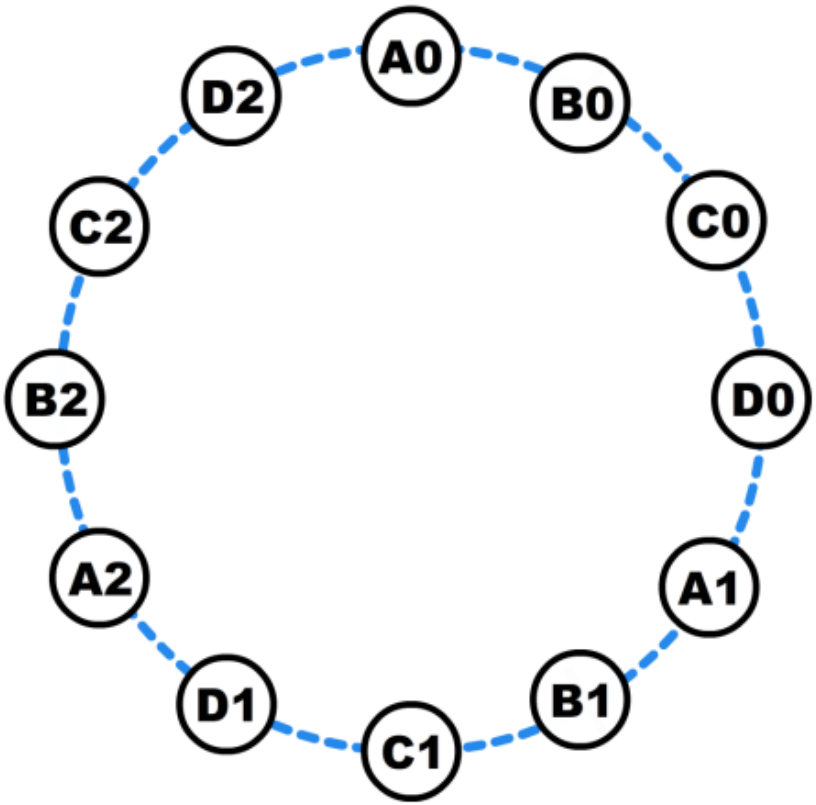
$$Q = 12 \mid M = \{A, B, C, D\} \rightarrow S = 4 \mid Q/S = 12/4 = 3$$



Partitionierung und Replikation (10)

Optimierte Partitionierungsstrategie (2)

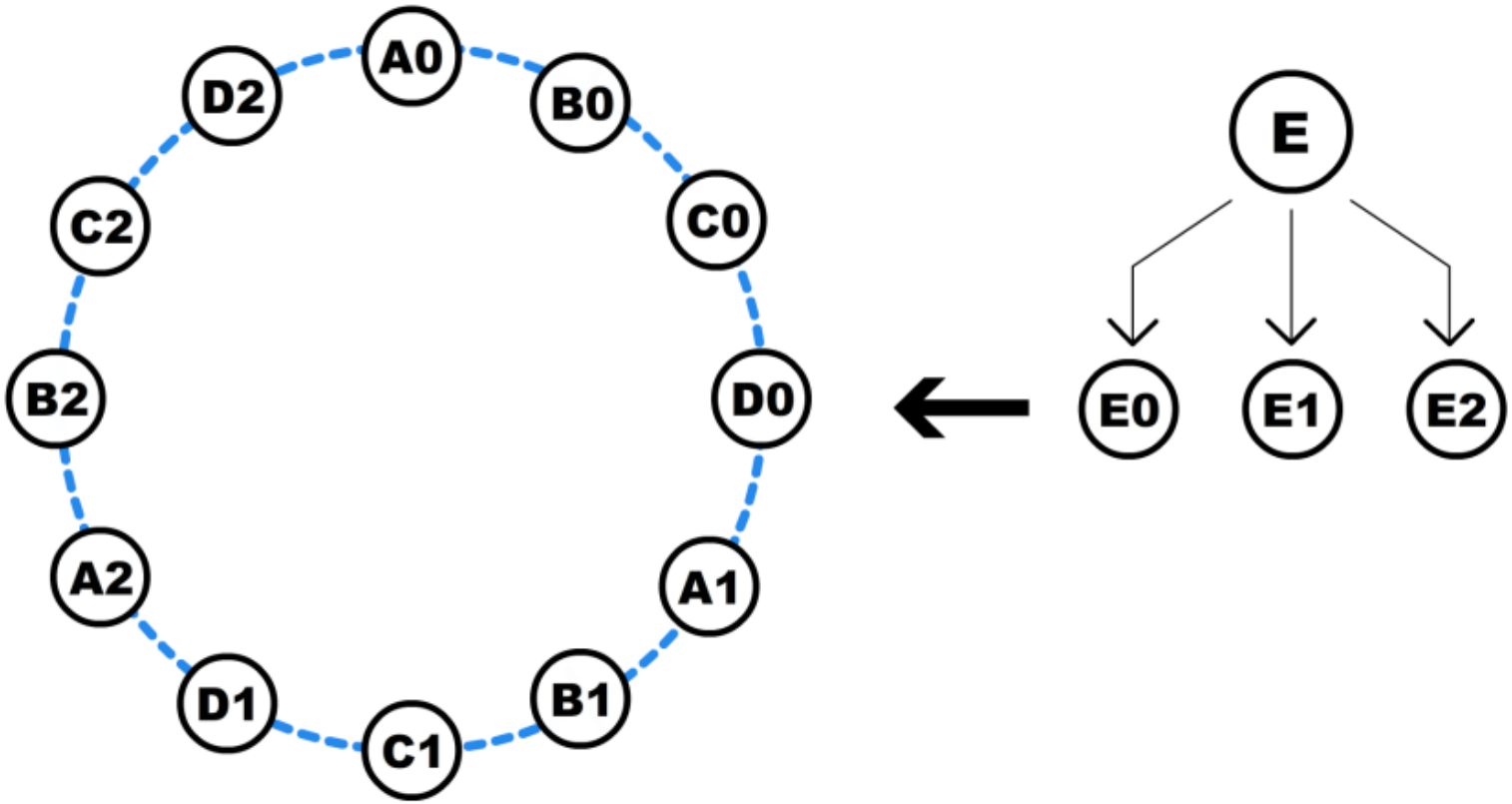
$$Q = 12 \mid M = \{A, B, C, D\} \rightarrow S = 4 \mid Q/S = 12/4 = 3$$



Partitionierung und Replikation (11)

Optimierte Partitionierungsstrategie (3)

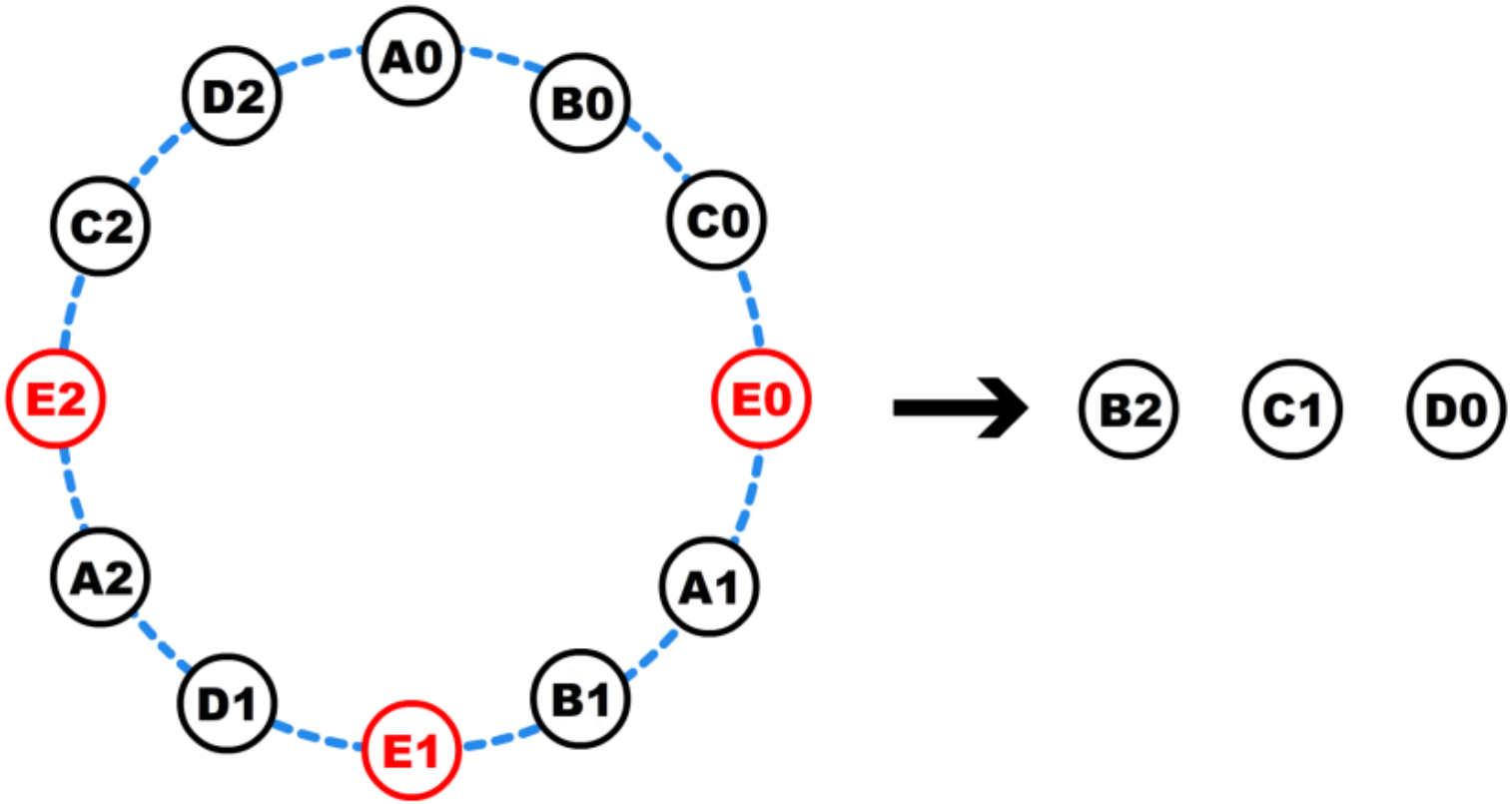
Hinzufügen des Knotens E; Q = 12 | S = **5** | Q/S = 12/5 = **2.4**



Partitionierung und Replikation (11)

Optimierte Partitionierungsstrategie (3)

Hinzufügen des Knotens E; Q = 12 | S = **5** | Q/S = 12/5 = **2.4**



Temporäre Ausfälle

Hinted Handoff

- Falls ein Knoten der Vorzugsliste ausfällt, Ausweichknoten auswählen **("Handoff")**
- Dieser speichert das Replikat in einer separaten Datenbank mit einem Hinweis **("Hint")** ab
- Der Hinweis verweist auf den ausgefallenen Knoten
- Rückgabe des Replikats an den zuvor ausgefallenen Knoten, wenn dieser wieder anläuft
- Periodisches Abfragen dieser Datenbank

Wiederherstellung bei permanenten Fehlern

Problem

- Ausweichknoten fallen auch aus
- Problematisch, wenn vorher die Replikate in der separaten Datenbank nicht „zurückgegeben“ wurden

Wiederherstellung bei permanenten Fehlern

Problem

- Ausweichknoten fallen auch aus
- Problematisch, wenn vorher die Replikate in der separaten Datenbank nicht „zurückgegeben“ wurden

Lösung

- Abgleich mit den Knoten der Vorzugsliste nach Wiederanlauf
- Ermitteln, ob und welche Replikate unterschiedlich sind

Versionierung

Letztendliche Konsistenz

- **Eventual Consistency**: Inkonsistente Zustände müssen irgendwann aufgelöst werden
- Entscheidend sind das „*Wann*“ und das „*Wer*“ der Auflösung
- **Wann?** Immer beim Lesen („always writeable“)
- **Wer?** Die Anwendung oder das System selbst
  $\Rightarrow$ Anwendungen müssen speziell daraufhin angepasst werden
- Die Entscheidung über den **Abgleich** erfolgt mittels **Vektoruhren**

Beispiel: Shopping-Cart-Service

- Basis-Operationen:
  - „*Zum Warenkorb hinzufügen*“: Darf niemals fehlschlagen oder verlorengehen
  - „*Vom Warenkorb entfernen*“: Darf fehlschlagen...

Konfigurierbarkeit – Sloppy Quorum (1)

Konfigurierbarkeit und Sloppy Quorum

- **Quorum**: Menge von Knoten, die im Kollektiv eine Entscheidung treffen
- **Sloppy**: Da Knoten jederzeit ausfallen können, herrscht keine strikte Mitgliedschaft
- **Quorum-Tripel**: (N, R, W), $W + R > N$
- R/W: Anzahl der Knoten, die einen Lesevorgang/Schreibvorgang bestätigen müssen, bevor dieser als Ganzes bestätigt wird
- Werte für N, R und W **frei konfigurierbar** $\rightarrow$ Stellschraube für Performance, Verfügbarkeit, Dauerhaftigkeit und Konsistenz

Konfigurierbarkeit – Sloppy Quorum (2)

Beispiel-Konfigurationen

- "High performance read/write Engine":
  - $R = 1, W = N$
  - $W = 1, R = N$
- **In Dynamo**: (3, 2, 2) üblich

Mitgliedsfindung und Ausfallerkennung (1)

Gossip-basiertes Protokoll

- Jeder Knoten pflegt eine Tabelle mit Mitgliedsknoten
- Abgleich erfolgt periodisch, indem mit zufällig ausgewählten Knoten Kontakt aufgenommen wird
- Hinzufügen/Entfernen von Knoten geschieht explizit über eine Administrationskonsole
- Problem der Wettlaufsituation:
  Knoten wissen zunächst noch nichts voneinander

Mitgliedsfindung und Ausfallerkennung (2)

Wettlaufsituation entschärfen

- Symmetrie-Bedingung lockern: **Seed-Knoten**
- Normale Knoten mit zusätzlichen Eigenschaften:
  - Registriert bei einem Discovery-Service (Erkennungsdienst)
  - Jedem anderen Knoten von Anfang an bekannt
- Administrator verbindet sich direkt zu einem Seed-Knoten
- Andere Knoten gleichen **letztendlich** ihre Liste mit diesen Knoten ab
- Problem der Wettlaufsituation wird entschärft

Mitgliedsfindung und Ausfallerkennung (3)

Ausfallerkennung

- Ausfallerkennung geschieht implizit bei der Kommunikation
- Wird keine Antwort erwidert, so wird angenommen, dass der kontaktierte Knoten ausgefallen ist $\rightarrow$ **Timeout-Prinzip**
- In regelmäßigen Abständen nach Wiederanlauf abfragen

Gliederung

Optimierungen (1)

Load Balancing

Eine Lese- oder Schreibanfrage gelangt erst über den Load Balancer zum passenden Knoten („extra network hop“)

Optimierungen (1)

Load Balancing

Eine Lese- oder Schreibanfrage gelangt erst über den Load Balancer zum passenden Knoten („extra network hop“)

Alternative

- Direkt in den Anwendungen Knoteninformationen speichern
- Anwendungen können eine Anfrage direkt zum passenden Knoten weiterleiten („zero-hop“)

Optimierungen (2)

Pufferung

- Einige Datenobjekte direkt im Hauptspeicher halten
- Lesevorgang überprüft erst den Inhalt des Puffers
- Verfügbarkeit (Performance) auf Kosten der Konsistenz weiter erhöhen (bei Ausfällen sind die Daten im Puffer weg!)

Optimierungen (2)

Pufferung

- Einige Datenobjekte direkt im Hauptspeicher halten
- Lesevorgang überprüft erst den Inhalt des Puffers
- Verfügbarkeit (Performance) auf Kosten der Konsistenz weiter erhöhen (bei Ausfällen sind die Daten im Puffer weg!)

Admission Controller („Einweiser“)

- Eine Art Scheduler (Einplaner) für die Hinter- und Vordergrundprozesse
- Legt die Anzahl der Zeitschlitze für die Hintergrundprozesse fest (z.B. Replikate synchronisieren, Mitgliedsfindung)
- Vordergrundprozesse haben höhere Priorität

Gliederung

Google File System (GFS)

Einige Besonderheiten von GFS

- **Zentralisierte** Koordination: Wenige Master-Server verwalten viele (Tausende) Chunkserver
- Commodity-Hardware
- Streaming-Access: WORM (write once read many)
- **Durchsatz** wichtiger als Latenz
- Partitionierung: Daten werden in Chunks aufgeteilt und auf die Chunkserver verteilt
- Replikation: Chunk-Kopien (Defaultwert: 3)

Cassandra (Apache)

Einige Besonderheiten von Cassandra

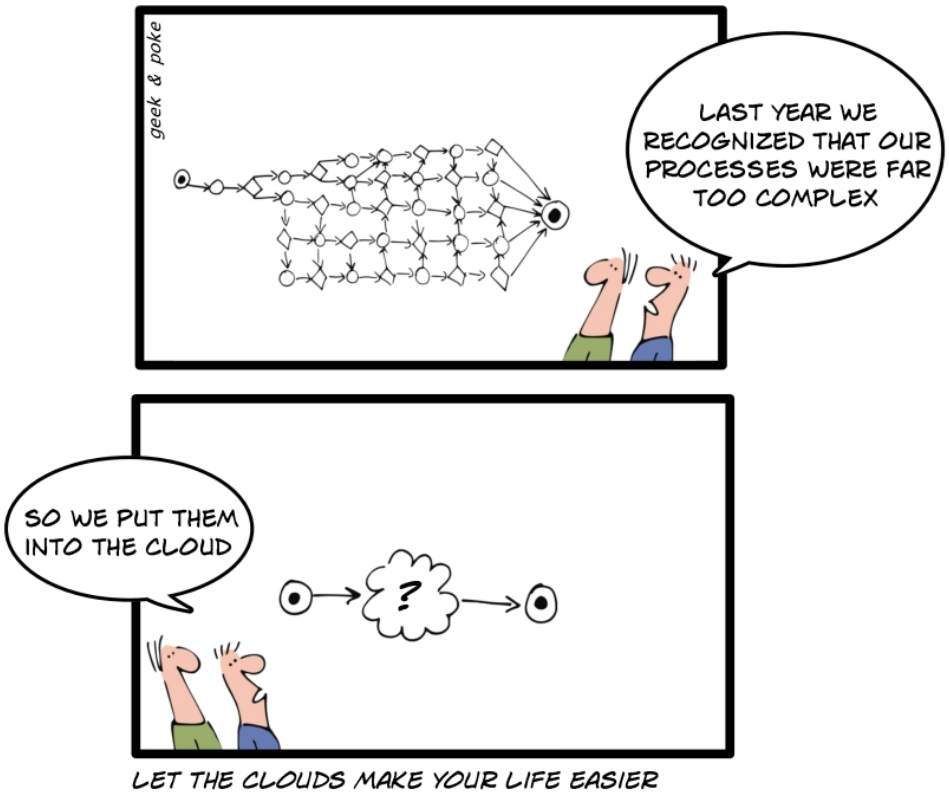
- **Motivation**: Inbox-Search für Nachrichten auf Facebook
- Starke Ähnlichkeit zu Dynamo
  (mind. ein Dynamo-Entwickler im Team)
- Hauptunterschied: **Semistrukturierte** Daten
- Operationen:
  - `insert(table, key, rowMutation)`
  - `get(table, key, columnName)`
  - `delete(table, key, columnName)`

Gliederung

Schlussbemerkung

Fazit

- Dynamo: Skaliert nicht endlos (**Schwachstelle**: Verwaltung der Knoteninformationen)
- Dynamo ist Grundlage für einige freie **Open-Source-Projekte** wie *Dynomite*, *riak* und *Voldemort*
- **Public Cloud**: Amazon S3, Speicherplatz kann gegen Bezahlung angemietet werden
- **Trend**: Weg von komplexen und teuren Lösungen (z.B. RAID) – hin zur Verwendung günstiger Commodity-Hardware



Quelle: [3]

The End

Vielen Dank
für eure Aufmerksamkeit!

Literaturverzeichnis

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels.
Dynamo: Amazon's Highly Available Key-value Store.
In *Proceedings of the 21st ACM Symposium on Operating Systems Principle*, pages 205–220, 2007.

`http://www.pro-datenrettung.net/fileadmin/bilder/`.

`http://geekandpoke.typepad.com/geekandpoke/cloud/`.