Consistent Hashing

Skalierbares Hashing für verteilte Systeme

David Heigl

January 21, 2026

Motivation

- Daten auf mehrere Server verteilen.
- Schneller Zugriff und gleichmäßige Last.
- Häufige Änderungen der Serveranzahl.

Klassisches Hashing

Idee:

$$N = \text{Anzahl der Server}$$

$$\text{Server-Index} = \text{hash(Key)} \bmod N$$

- Einfach und effizient.
- Problematisch bei Änderung von N (z.B. Server hinzu oder Ausfall).

Problem des klassischen Hashings

- Server fällt aus oder kommt hinzu.
- Fast alle Keys werden neu zugewiesen.
- Hoher Netzwerk- und Rechenaufwand.

Klassisches Hashing: Beispiel

Beispiel: Erhöhung von $N = 3$ auf $N = 4$

$$\text{Server-Index} = \text{hash(Key)} \bmod N$$

Key	$N = 3$	$N = 4$	**Ziel-Server**
10	$10 \bmod 3 = 1$	$10 \bmod 4 = 2$	geändert
20	$20 \bmod 3 = 2$	$20 \bmod 4 = 0$	geändert
30	$30 \bmod 3 = 0$	$30 \bmod 4 = 2$	geändert
40	$40 \bmod 3 = 1$	$40 \bmod 4 = 0$	geändert

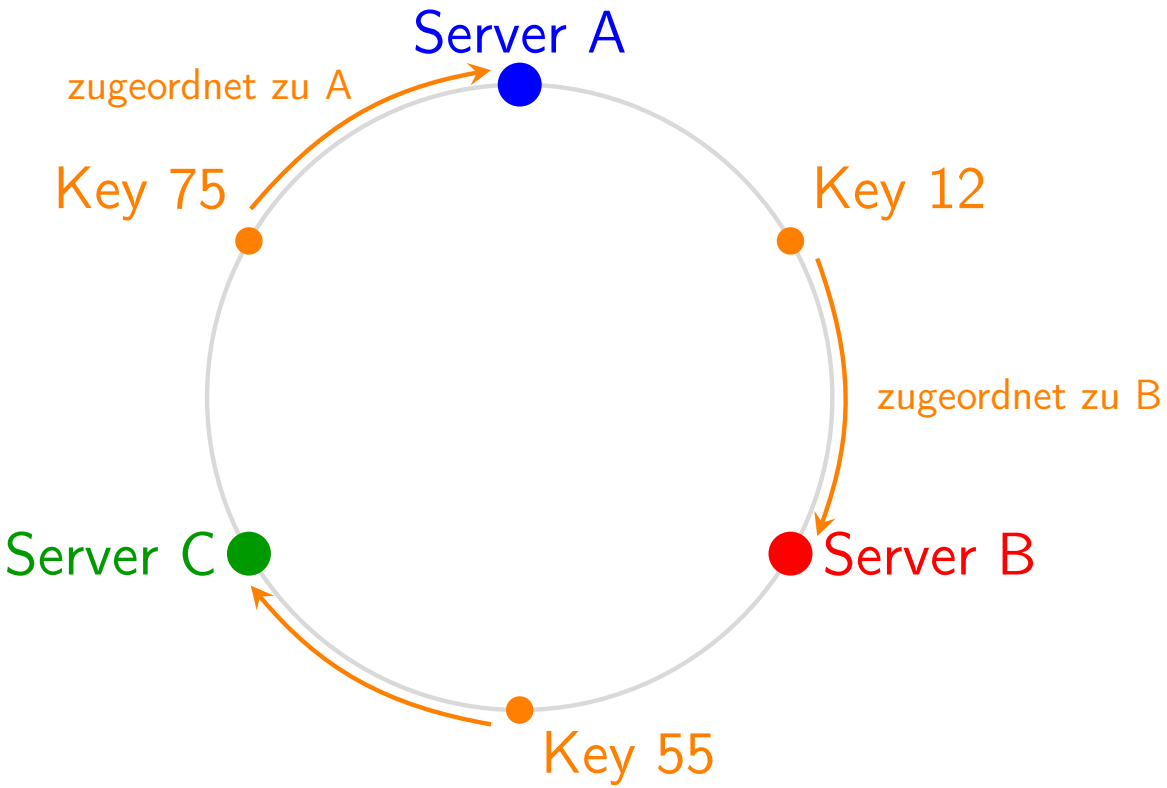
- Fast jeder Key wird einem neuen Server zugewiesen.

Grundidee von Consistent Hashing

- Hashraum als Kreis (Ring).
- Server und Keys werden gehasht.
- Zuordnung zum nächsten Server im Uhrzeigersinn.

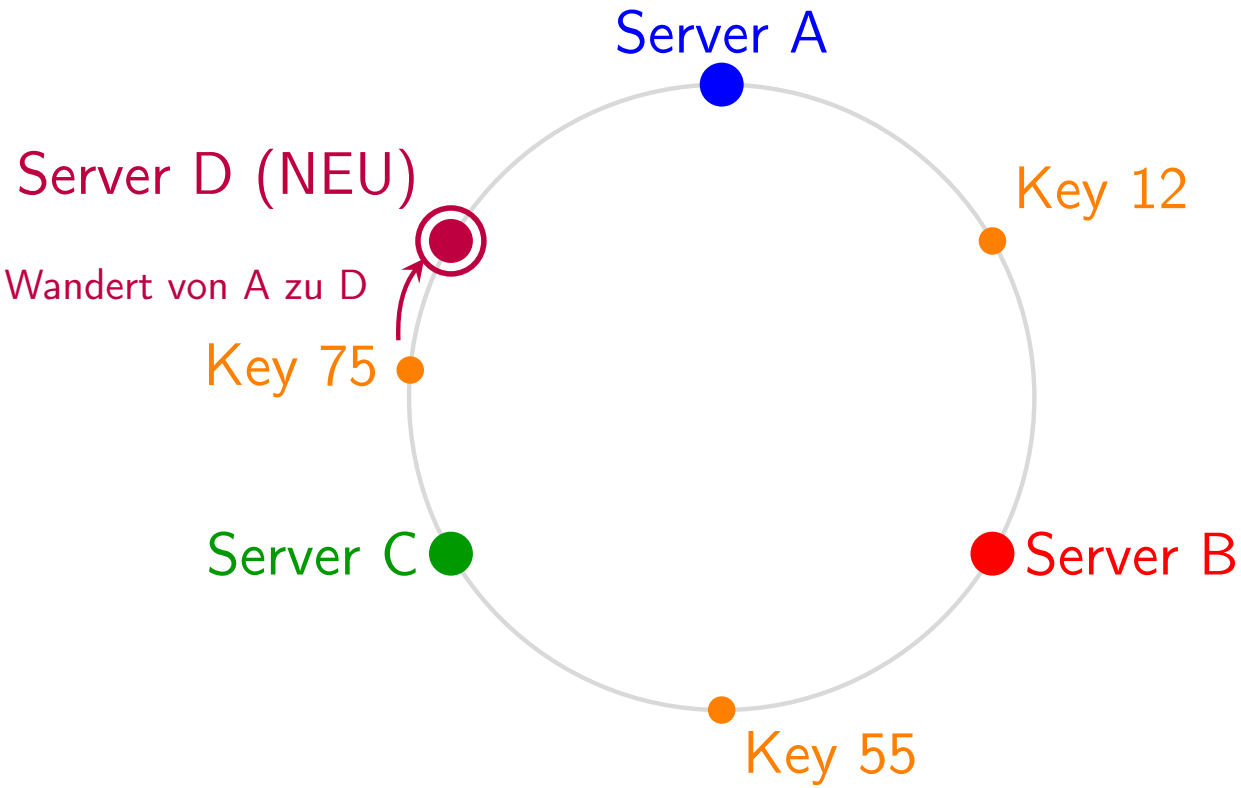
Karger et al., "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web"

Ring-Darstellung



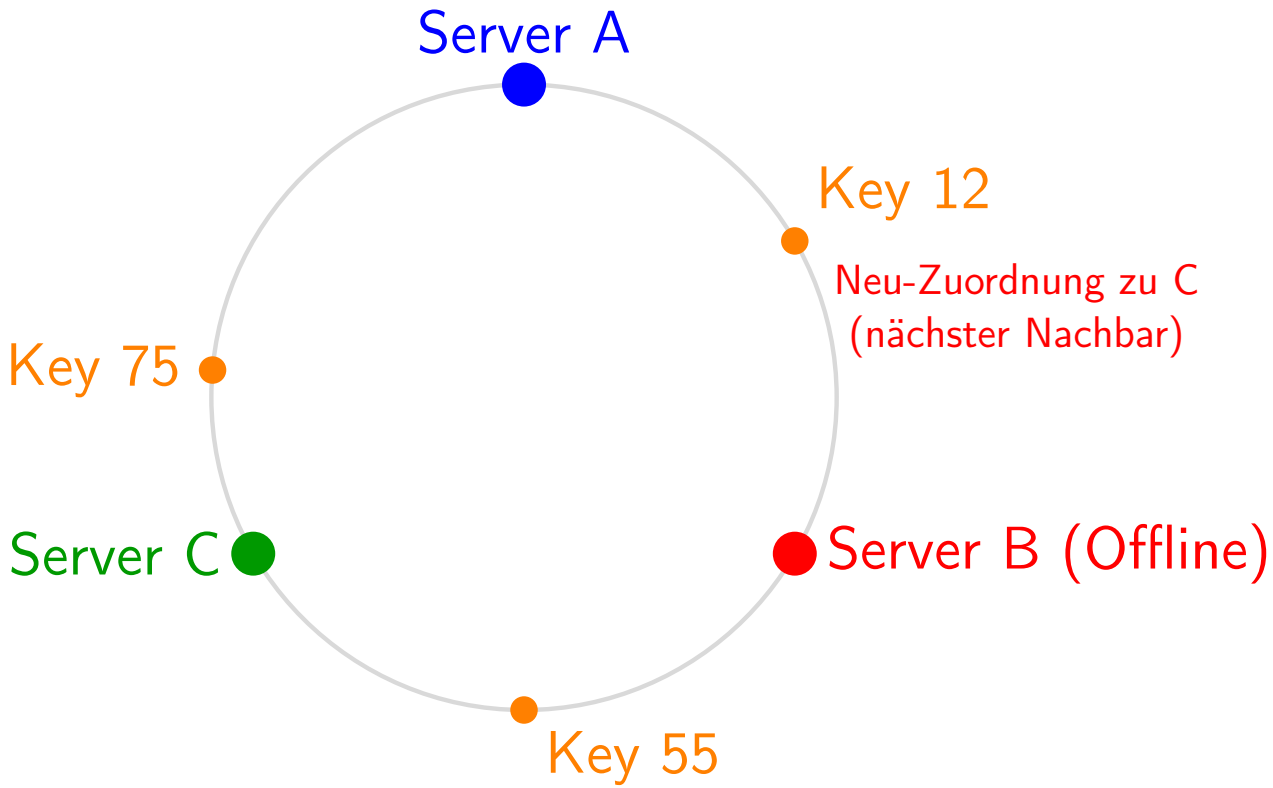
- Keys werden dem nächsten Server im **Uhrzeigersinn** zugewiesen.
- Hash-Bereich (z.B. 0 bis N) wird auf 0° bis 360° abgebildet.

Hinzufügen eines Servers



- Nur der Bereich zwischen Server C und D wird neu zugeordnet.
- Keys in allen anderen Abschnitten (A-B, B-C) bleiben unberührt.

Server-Ausfall



- Server B ist nicht mehr erreichbar.
- Anfragen für Keys im Bereich (A bis B) landen automatisch beim nächsten Knoten im Ring (**Server C**).

Warum ist das besser?

- Nur ca. $1/N$ der Keys werden neu verteilt.
- Weniger Datenbewegung.
- Gute Skalierbarkeit.

Problem: Ungleichmäßige Last

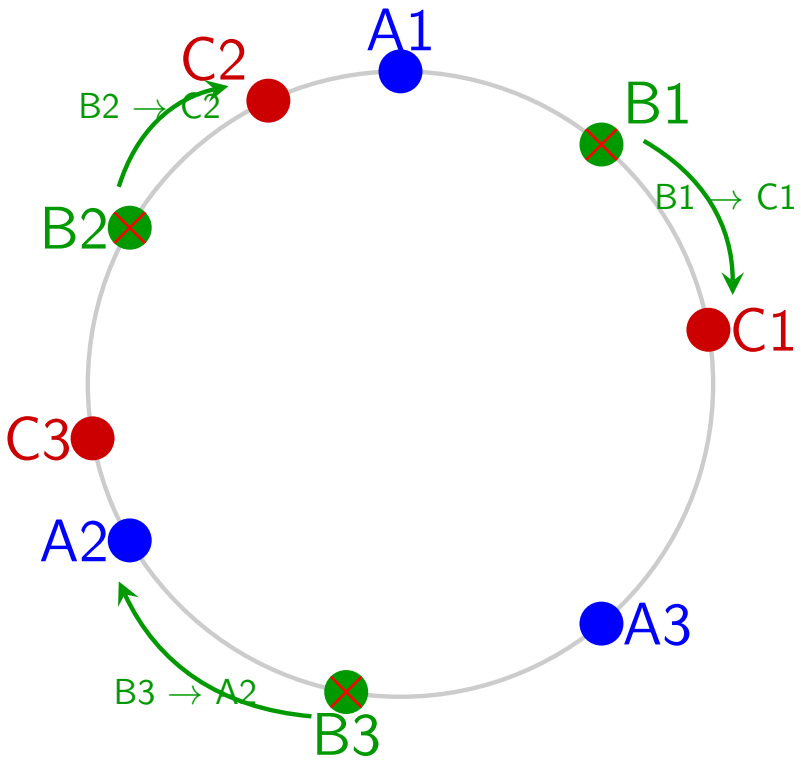
- Zufällige Serverpositionen.
- Manche Server erhalten mehr Keys.

Virtuelle Knoten

- Jeder Server erscheint mehrfach auf dem Ring.
- Gleichmäßigere Lastverteilung.
- Bessere Ausfallsicherheit.

Stoica et al., "Chord: A scalable peer-to-peer lookup service for internet applications"

Virtuelle Knoten: Lastverteilung beim Ausfall



- Bei Ausfall von **Server B** werden die Lasten auf **A** und **C** verteilt.
- Jeder Server übernimmt nur einen Teil der Last von B.

Vorteile und Nachteile

Vorteile

- Skalierbar.
- Robust gegenüber Änderungen.

Nachteile

- Höhere Komplexität.
- Verwaltungsaufwand für Ring.

Praxisbeispiele

- Amazon Dynamo.
- Apache Cassandra.
- Redis / Memcached.

DeCandia et al., “Dynamo: Amazon’s highly available key-value store”

Vereinfachte Erklärung

- Kreis mit Ablageplätzen.
- Jeder Gegenstand wird dem nächsten Platz im Uhrzeigersinn zugewiesen.
- Neuer Platz oder Ausfall: nur wenige Gegenstände müssen umverteilt werden.

Fazit

- Löst ein zentrales Problem verteilter Systeme.
- Minimale Datenbewegung.
- Industriestandard.

Quellen

- DeCandia, Giuseppe et al. "Dynamo: Amazon's highly available key-value store". In: *SIGOPS Oper. Syst. Rev.* 41.6 (Oct. 2007), pp. 205–220. ISSN: 0163-5980. DOI: 10.1145/1323293.1294281. URL: https://doi.org/10.1145/1323293.1294281.
- Karger, David et al. "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web". In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*. STOC '97. El Paso, Texas, USA: Association for Computing Machinery, 1997, pp. 654–663. ISBN: 0897918886. DOI: 10.1145/258533.258660. URL: https://doi.org/10.1145/258533.258660.
- Stoica, Ion et al. "Chord: A scalable peer-to-peer lookup service for internet applications". In: *SIGCOMM Comput. Commun. Rev.* 31.4 (Aug. 2001), pp. 149–160. ISSN: 0146-4833. DOI: 10.1145/964723.383071. URL: https://doi.org/10.1145/964723.383071.