Consistent Hashing

Consistent Hashing হলো একটি বিশেষ পদ্ধতি যা ডিস্ট্রিবিউটেড সিস্টেমে (যেমন: অনেকগুলো ক্যাশ সার্ভার বা ডাটাবেস) ডাটাগুলোকে এমনভাবে ভাগ করে রাখে যাতে কোনো নতুন সার্ভার যুক্ত হলে বা পুরনো সার্ভার নষ্ট হয়ে গেলে খুব সামান্য পরিমাণ ডাটা নাড়াচাড়া করতে হয়।

সাধারণ হ্যাশিং পদ্ধতিতে যদি একটি সার্ভার কমে বা বাড়ে, তবে প্রায় সব ডাটার জায়গা পরিবর্তন হয়ে যায়, যা সিস্টেমের জন্য বিশাল ঝামেলার কারণ। Consistent Hashing এই সমস্যা সমাধান করে।

১. সাধারণ হ্যাশিং (Traditional Hashing) এর সমস্যা

ধরুন আপনার কাছে ৩টি সার্ভার আছে (S_0, S_1, S_2) এবং আপনি ডাটাগুলো এভাবে সেভ করছেন: server = hash(key) % n (যেখানে n হলো সার্ভারের সংখ্যা)।

- এখন যদি $n=3$ হয় এবং আপনি একটি নতুন সার্ভার যোগ করেন ($n=4$), তবে মেজরিটি ডাটার লোকেশন বদলে যাবে।
- এর ফলে আপনার সব **Cache** ইনভ্যালিড হয়ে যাবে এবং সার্ভারে হঠাৎ বিশাল চাপ পড়বে (একে বলে **Cache Stampede**)।

২. Consistent Hashing কীভাবে কাজ করে?

Consistent Hashing এই সমস্যা সমাধানে একটি কাল্পনিক বৃত্ত বা **Hash Ring** ব্যবহার করে।

ক. দ্য হ্যাশ রিং (The Hash Ring)

মনে করুন, একটি বৃত্তের পরিধি বরাবর ০ থেকে $2^{32}-1$ পর্যন্ত সংখ্যা আছে। এই বৃত্তের বিভিন্ন জায়গায় আমরা আমাদের সার্ভারগুলোকে বসাই।

খ. ডাটা বা কি (Key) বসানো

একইভাবে, আমরা আমাদের ডাটা বা অবজেক্টগুলোকেও ওই একই বৃত্তে হ্যাশ করে বসাই। এখন প্রশ্ন হলো—কোন ডাটা কোন সার্ভারে যাবে?

- **নিয়ম:** প্রতিটি ডাটা তার অবস্থান থেকে ঘড়ির কাঁটার দিকে (**Clockwise**) ঘুরতে থাকবে এবং সামনে প্রথম যে সার্ভারটি পাবে, সেখানেই সেভ হবে।

গ. সার্ভার যোগ বা বিয়োগ করলে কী হয়?

- **সার্ভার রিমুভ করলে:** যদি একটি সার্ভার নষ্ট হয়ে যায়, তবে শুধু ওই সার্ভারের ডাটাগুলো তার পরের সার্ভারে চলে যাবে। বাকি সব সার্ভারের ডাটা আগের মতোই থাকবে।
- **সার্ভার যোগ করলে:** নতুন একটি সার্ভার বসালে শুধুমাত্র তার পেছনের কিছু ডাটা নতুন সার্ভারে আসবে। পুরো সিস্টেমের ডাটা পাল্টাতে হবে না।

৩. ভার্চুয়াল নোড (Virtual Nodes)

কখনও কখনও রিং-এ সার্ভারগুলো এমনভাবে বসে যে একটি সার্ভারের ওপর অনেক বেশি ডাটা চলে আসে আর অন্যটি খালি থাকে। একে **Data Skew** বলে।

- **সমাধান:** প্রতিটি ফিজিক্যাল সার্ভারের অনেকগুলো **Virtual Nodes** বা নকল কপি রিং-এর বিভিন্ন জায়গায় বসিয়ে দেওয়া হয়। এতে ডাটা সব সার্ভারে সমানভাবে ডিস্ট্রিবিউট হয়।

৪. Consistent Hashing-এর সুবিধা

1. **Scalability:** খুব সহজে সার্ভার বাড়ানো বা কমানো যায়।
2. **Minimal Data Movement:** সার্ভার পরিবর্তনের ফলে খুব কম পরিমাণ ডাটা রি-হ্যাশ করতে হয়।
3. **High Availability:** ডিস্ট্রিবিউটেড ক্যাশিং (যেমন: Redis, Memcached) এবং লোড ব্যালেন্সিংয়ের জন্য এটি অপরিহার্য।

৫. কোথায় ব্যবহার করা হয়?

- **Amazon DynamoDB** এবং **Apache Cassandra** ডাটাবেসে।
- **Content Delivery Networks (CDN)** যেমন Akamai বা Cloudflare-এ।
- ডিস্ট্রিবিউটেড ক্যাশিং সিস্টেমে।

সংক্ষেপে সারমর্ম:

Consistent Hashing হলো ডাটা ডিস্ট্রিবিউশনের এমন এক বুদ্ধিমান পদ্ধতি যা রিং বা বৃত্তাকার ম্যাপ ব্যবহার করে নিশ্চিত করে যে সার্ভার সংখ্যা কমলে বা বাড়লে যেন পুরো সিস্টেম ওলটপালট না হয়ে যায়।

🧠 Problem First (Why Consistent Hashing?)

Normal Hash Sharding

shard = hash(key) % N

❌ Problem:

- N change হলে (add/remove shard)
- **Almost all keys move**
- Massive rebalancing 😵

📌 This is unacceptable at scale.

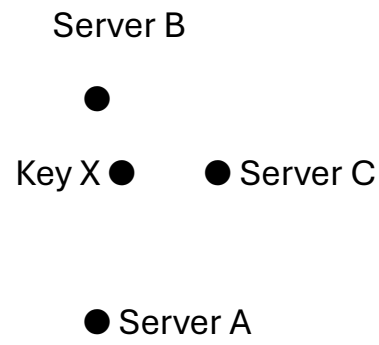
🧠 Consistent Hashing কী?

👉 **Keys & servers both placed on a hash ring (circle)**
👉 Each key goes to the **next clockwise server**

0 ——————————— 360° (hash space)

📌 Adding/removing server → **only small subset of keys move**

🔴 Hash Ring Concept (Core)



Rule:

A key is assigned to the **first server clockwise**

1. How It Works (Step-by-Step)

Step 1: Hash servers

hash(ServerA) → position on ring

hash(ServerB) → position on ring

Step 2: Hash keys

hash(userId)

Step 3: Assign key

Key → nearest clockwise server

2. Add a New Server (MAGIC PART ✨)

Before

A ———— B ———— C

After adding D

A ——— D ——— B ——— C

📌 Only keys between **A → D** move
❌ Not all keys

📌 Interview line:

“Consistent hashing minimizes data movement during re-sharding.”

3. Remove a Server

If B fails:

Keys of B → move to next clockwise server

✓ Automatic failover
✓ Minimal reshuffle

4. Virtual Nodes (VERY IMPORTANT)

Problem:

- Uneven load if few servers

Solution:

👉 **Virtual nodes (vnodes)**

Server A → A1, A2, A3

Server B → B1, B2, B3

✓ Better distribution
✓ Load balancing

📌 Interview line:

“Virtual nodes improve uniformity in consistent hashing.”

5. Consistent Hashing vs Normal Hashing

Aspect	Normal Hash	Consistent Hash
Shard change	Full reshuffle	Minimal movement
Scaling	Painful	Smooth
Complexity	Simple	Medium
Used in	Small systems	Large-scale systems

6. Where Is It Used? (REAL SYSTEMS)

- DynamoDB
- Cassandra
- Kafka partitioning
- Load balancers
- CDN routing

📌 FAANG line:

“Consistent hashing is widely used in distributed storage and caching systems.”

7. Consistent Hashing + Replication (Advanced)

Each key mapped to:

Primary server + next N replicas

✓ High availability
✓ Fault tolerance

8. When NOT to Use Consistent Hashing?

- Small DB
- Static shard count
- Heavy range queries

🏆 FAANG Interview Perfect Answer 💎

Use this 👇

“Consistent hashing distributes keys across a hash ring such that adding or removing nodes causes minimal data movement. With virtual nodes, it ensures even load distribution and is widely used in scalable distributed systems like DynamoDB and Cassandra.”

🧠 One-Line Memory Hook

Consistent hashing = scale without reshuffle