Armen Melkumyan • 1st
Technical / Solutions Architect
1w •

Latency vs. Throughput:

In today's understanding the difference between latency (how fast you respond) and throughput (how much you handle) can make or break your system's performance. Here's a quick breakdown:

1️⃣ What is Latency?

Definition: The time from sending a request to receiving a response (measured in ms or μs).

Key Components:

Network Latency: Time to travel across networks (e.g., user in Australia -> U.S. server).

Processing Latency: Time the server/code takes to process requests.

Database Latency: Time to query/update the database.

Disk I/O Latency: Reading/writing from storage (HDD vs. SSD vs. in-memory).

Application Latency: Delays from inefficient code or dependencies.

End-to-End Latency: Sum of all delays above.

2️⃣ What is Throughput?

Definition: The number of requests/transactions a system can handle per second (RPS, QPS, TPS, MPS).

Key Metrics:

Requests Per Second (RPS): API calls processed/second.

Queries Per Second (QPS): Database queries executed/second.

Transactions Per Second (TPS): Completed transactions/second.

Messages Per Second (MPS): Processed messages/second.

⚠️ Important: High throughput doesn't automatically mean great performance if your latency is still high.

3️⃣ The Trade-off: Optimizing for Latency vs. Throughput

Online Gaming: Low latency (<10ms) is vital, throughput is less critical.

Stock Trading: Sub-millisecond speed is key; throughput is secondary.

Streaming Services: Slight latency in loading is okay, focus on serving millions concurrently (high throughput).

E-commerce Checkout: Fast response (<100ms) for user satisfaction, throughput is less priority.

Big Data Processing: High throughput to process large datasets, higher query latency is acceptable.

Key Insight: Real-time systems → optimize for low latency.

Large-scale distributed systems → optimize for high throughput.

4️⃣ Techniques to Optimize Both

For Low Latency

Use CDNs, Edge Computing, and optimized network routes.

Cache (Redis, Memcached) to reduce DB hits.

Minimize API call chains, use faster serialization (e.g., Protocol Buffers).

Proper indexing, read replicas, and denormalized DB structures.

#SystemDesign

Armen Melkumyan and 33 others 2 comments