System Design @ Scale

Timeout, Retries And Backoff

"Everything fails, all the time."

Motivation → It will not be possible to avoid failures by 100%, strive for systems to have mechanism in place to handle issues and minimise impact.

Increasing System Availability

① Timeouts

↳ add timeout to any remote call.

connection & request timeout.

what should be timeout value?

not to small not too large

↳ relative terms & totally dependent on system.

① Backend Traffic & latency ↑

② Too many retries → complete outage.

① resource consumption ↑

How to choose?

① p99 latency of dependent services.

↳ consider network latency as well.

↳ latency is closer to p50 → add small padding.

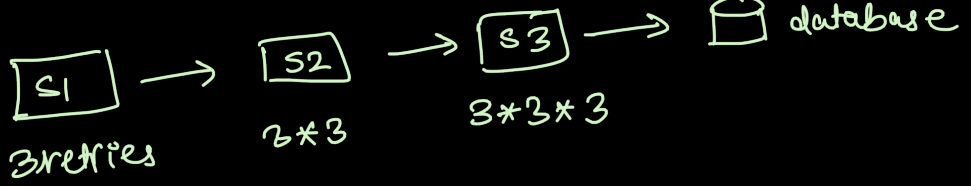
< In short, careful analysis of system is necessary to come up with timeout values >

② Retries and Back Off

To many retries can cause system overload. Though retries are helpful in resolving Transient issues.

Potential Problems

① Retries at multiple layers in distributed system can cause extensive load on end system.



② Circuit Breakers → Calls to downstream is stopped if threshold exceeds.
↳ difficult to test & adds time to recovery
↳ better solution could be token buckets.

③ when to Retry ? Idempotent api are safe to retry.
API with any side-effects are not.
↳ which failures to retry for? → not client errors
→ servers errors maybe

Back Off → Retries should be executed with back off.
wait for some time then retry

exponential back off → wait time increased exponentially after every attempt.
till some maximum attempt.

Jitter → lets say all client calls failed.
all with retry with Exponential backoff.

eventually it will hit system at same time ⇒ load ↑

Jitter adds randomness
spread the retries around time.

⇒ to make debugging easier, randomness can be introduced per server → example take server IP as input to rand().

See you again with new deep dive.

Stay Tuned on YouTube – MsDeep Singh

Happy Learning