#ASLI ENGINEERING

Handling Timeouts in Microservices

SWIPE

BY

ARPIT BHAYANI

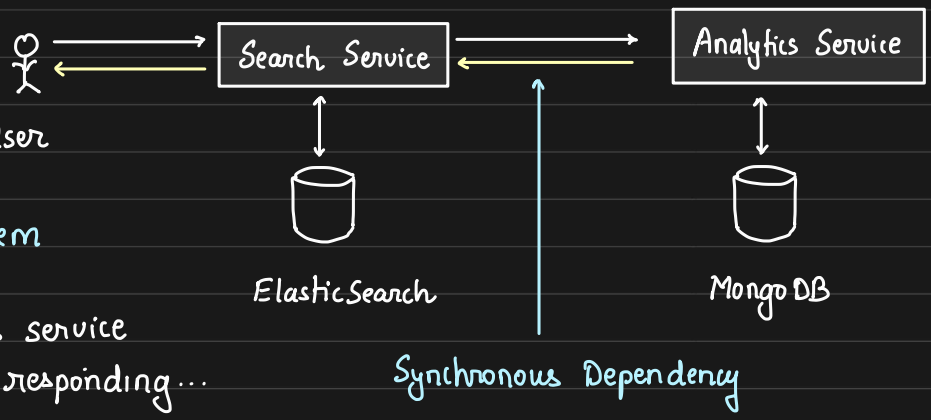
Handling timeouts in Microservices

Microservices are Great!

- Separation of concern
- Flexibility
- Picking the best tech stack
- Focussed Scaling

But there are concerns, one of them is Timeouts

Scenario: User wants to search Blogs. Search service computes most relevant blogs. Then talks to Analytics service to get Blog Views and then returns the response to the user.

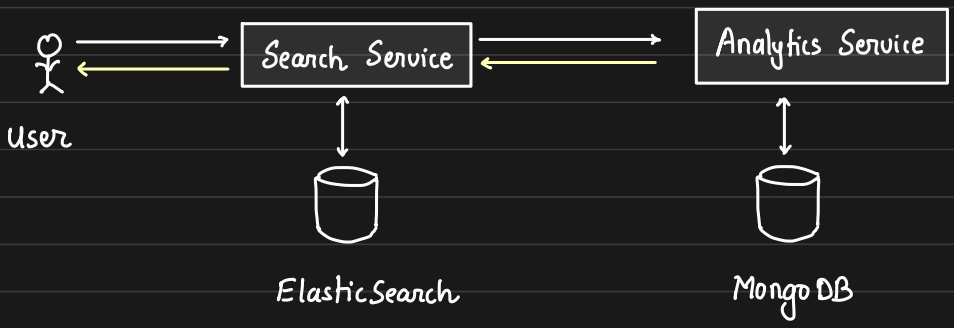


The problem

analytics service is not responding...

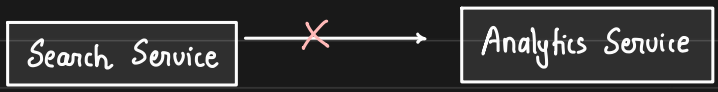
How long should you wait?
What action should you take?

Send partial response
Not send any response to the user

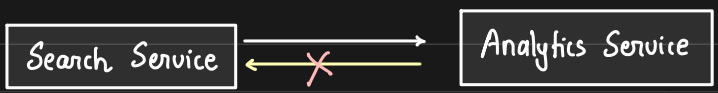


What could go wrong?

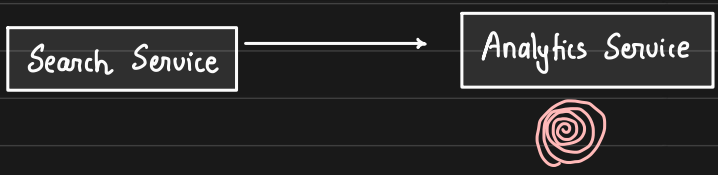
↳ Analytics service never got the request



↳ Response from the Analytics team never reached the Search Service



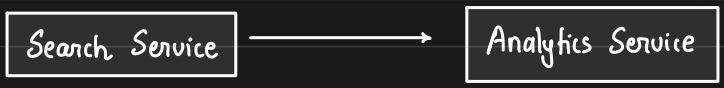
↳ Analytics service is taking too long to process



The core problem: Delays are arbitrarily long

So, how long do we wait? We cannot wait forever!

Use Timeouts, always



[timer]

Waits for the response from the analytics service for at the max 10 seconds.

↖ Timeout

How to handle timeouts, then....

Approach 1: Ignore [not recommended]

We assume operation succeeded → leads to unpredictable UX
but it actually failed...

Good practice: Catch all exceptions... everytime and then take an informed call depending on the context.

Approach 2: Configure and use defaults

Upon timeout, you may choose to use a default value

Approach 3: Retry

Assume that the remote operation failed, and retry.

Retries are simple when it is a "read request"
But, sometimes situation becomes tricky

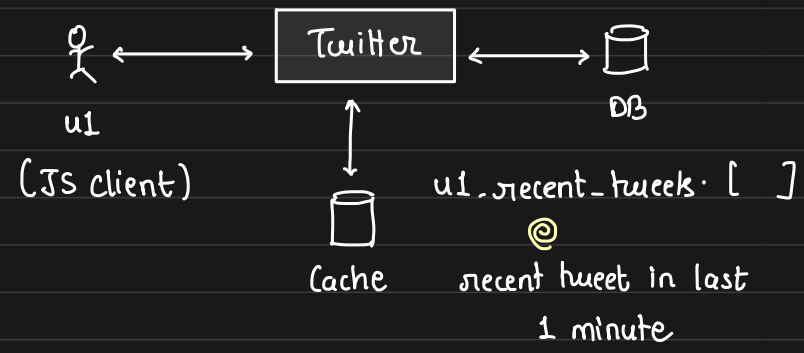
- ↳ request is non-idempotent eg: moving money from A to B
- ↳ request is expensive eg: heavy analytics query
- ↳ other service is overloaded and you add more load with retry

Good to have: Retries with exponential backoffs
[1s, 2s, 4s, 8s, ...]
Make services as idempotent as possible

Approach 4: Retry only if needed

In some cases, we may check for the completion and then decide to retry

eg: User tweeting the same post twice accidentally (within one minute)



Approach 5: Rearchitect

Event driven arch.

Remove synchronous communication wherever possible

Highlights:

always have timeouts

picking timeout value is tricky → Too short → false positives
→ Too long → perf. bottlenecks

make retries safe
↳ Idempotence