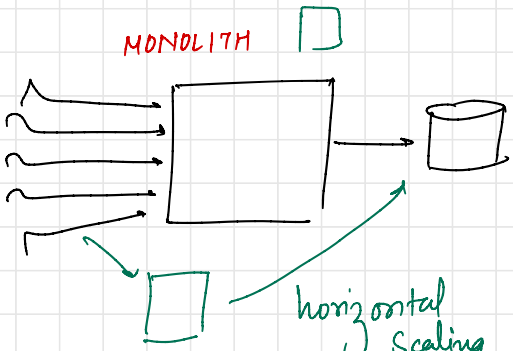
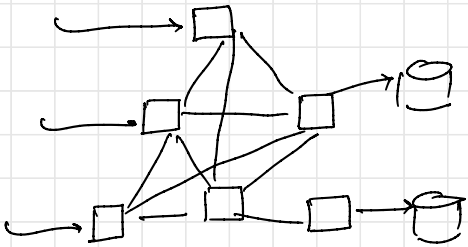
Monolith service and Microservices

MONOLITH



MICROSERVICE



* single business unit
* separate out services

Advantages

* Cohesive and compact team
* Easier to scale
* Less complexity
* Less Duplication
* Faster comparatively

Disadvantages

* More contex required for a new member
* Deployment are complicated
* Single point of failure

Advantages

→ Easier to scale database, adding new component easy
→ easy for new developers
→ work in parallel
→ scale server

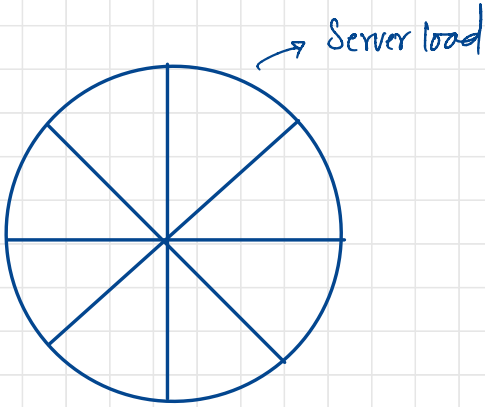
Disadvantages

→ harder to design
→ needs skilled architects

DATABASE SHARDING

Horizontal partitioning – Break data into peices & allocate to servers (use 1 key)

Vertical partitioning – uses columns



Sharing – Taking one attribute in the data and partitioning the data so that each server gets one chunk (DB Server)

↳ consistency ↳ Availability

What to Shard on?

College Data → USER ID

TINDER → Location (all users in same location)

Indexing within shard using different type

Problems :-

1) JOINS across SHARDS
2) FIXED num of shards
   ↳ divide shard – hierachial sharding to remove inflexibility
   ↳ for feilure
      ↳ master – slave arch

DISTRIBUTED CACHING

WHY NEEDED?

→ Saving multiple network calls [key : value]

→ avoid load on database

→ replace expensive computations

* Cache policy (IMP) → data loading to & removing from cache
  ① Least Recently used // on top of cache
    ↳ Remove last
  ② Least Frequently used

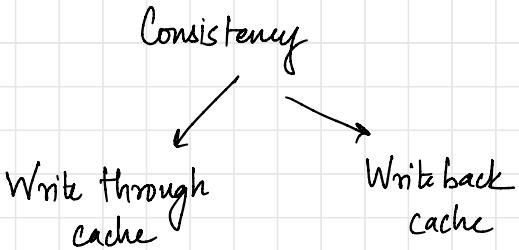
Issues with bad policy

① Expensive extra calls ② Thrashing ③ Consistency

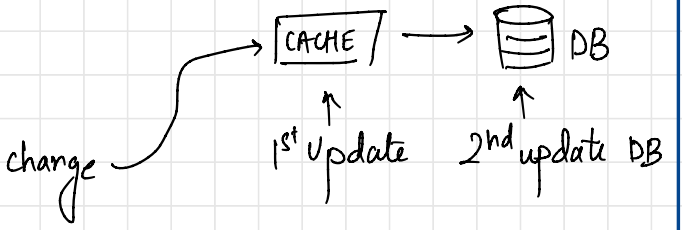
Where to place cache memory ? [Redis]

Global cache # slightly slower than cache at main database (local)

Higher Accuracy



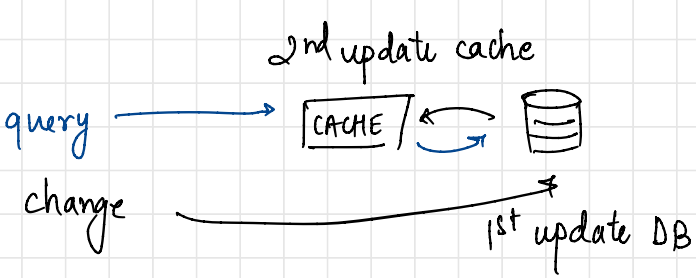
THROUGH



ISSUE

multiple servers consistency problem

WRITE



ISSUE

Thrashing Server

expensive multiple changes

HYBRID MODEL

for list of non-critical changes, update cache not server

list of changes (persistent) → update server in bulk