Designing Data Intensive Applications

CHAPTER 1

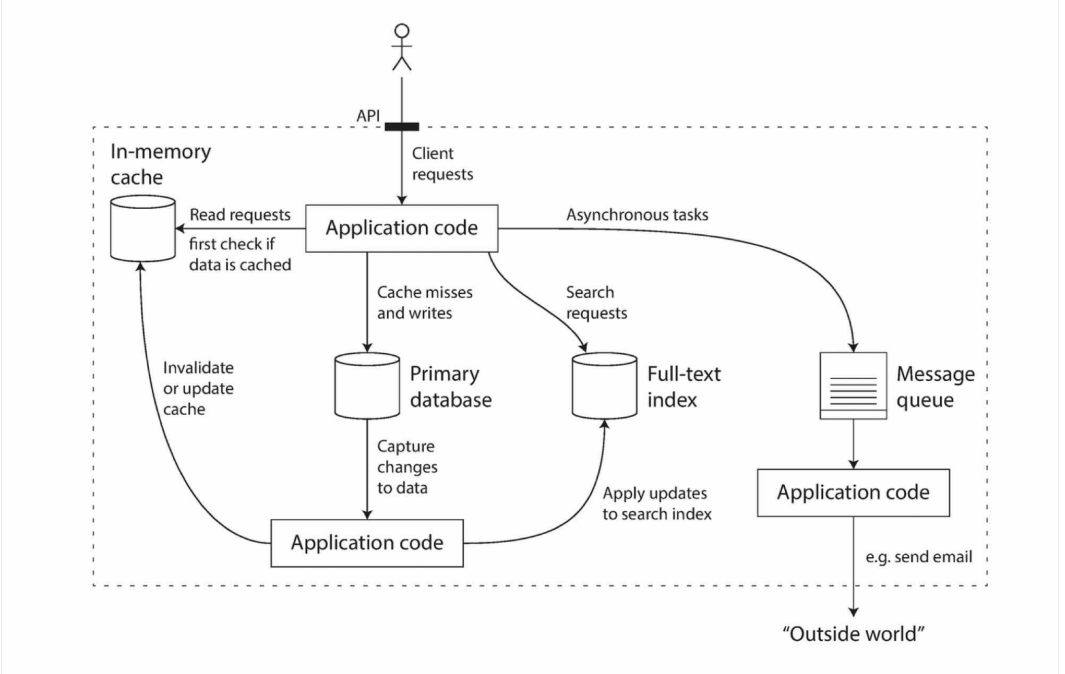
Reliable, Scalable and Maintainable Applications

Reliability → Tolerating Hardware and Software failures
Human Error.

Scalability → Measuring load and performance
Latency percentiles, throughput

Maintainability → Operability, Simplicity and evolvability

One Possible Architecture



How will you ensure below when designing a service →

① Ensure data is correct even if things go wrong?

② Provide good performance to clients even when parts of system are degraded?

③ Handle sudden increase/burst in TPS?

④ How will you design an API for system?

Points to Consider →

① There is no single solution to solve a problem.
It could be possible that ElasticSearch is best search indexing solution available but you went ahead with Solr due to legacy dependencies.

Design can depend on →

① Skills and experience of people.

② legacy system dependencies.

③ time scale for delivery.

④ Organization's tolerance for different kind of risks.

⑤ Regulatory constraints.

Three Important pillars of Software System →

[A] Reliability

System should continue to work correctly even with hardware/software faults or Human errors.

with expected traffic.

What can go wrong in system ?

These are called faults.

→ Fault is different from failures.

↳ Watch Error vs Fault vs Failure video on my youtube channel.

↳ Fault means server was unable to serve your request. (probably a specific component of service)

↳ Failure means whole system stopped providing required service to user.

↳ To make systems Fault Tolerant, you can design experiments to test system limits.

↳ Watch "Chaos Engineering" video on channel.

Hardware Faults →

any issues in system Hardware – CPU, disk, RAM, router ....etc.

Example → On storage cluster of 10000 disks, avg 1 disk die/day.

To solve this, you should have backups in place

↑ if one component dies, redundant component can take its place.

Software Errors →

Any issue in software which is running on hardware.

Example –

① Bug in software

② You can try to make your system fault tolerant, but if there are issues in third party software our application use, the system can get into failure state.

③ The upstream service is unresponsive.

④ Cascading failures – fault in one component triggers fault in another component

↑ watch "cascading failures" video on YT channel.

Suggestive Resolutions

① Try to close on all unknowns / edge cases during System design.

② Testing

③ Process isolation
↳ example tabs in Chrome browser.

④ Allow process to crash and restart

⑤ Monitoring + chaos engineering + Alerting + Logging

Human Error → Humans are unreliable ☹

① Experiments in production should be well monitored.

② Right amount of abstraction.

③ Thorough Testing.

④ Monitoring + Alerting. + Best training practices.

So what do you think?

Is Reliability Important?

Scalability → what happens if your system's load is increased by 10x, will it be reliable?

> It is used to describe system's capability to handle increased load on system.

what is load? → TPS for web server, read write ratio on database, hit rate on cache..

Example Twitter → How a tweet published by user reach his/her followers?

↳ followers asks for new tweet?

↳ user pushes tweet to all its followers?

⇒ Watch similar analogy in "Instagram NewsFeed" video on YouTube.

Here in case of twitter, distribution of followers per user is key load parameter to think around Scalability.

Performance → Is system working as per our expectations?

→ What is TPS that can be handled by server for given CPU and memory?

→ What is latency to serve API response?

Response Time determination → p50, p90, p99, etc.

lets say response time = 1 second, then p90 means 90 out of 100 requests take <1sec and other 10 requests take 1sec or more.

How to handle LOAD? load is increased but performance is not impacted.

- → vertical scaling
- → Horizontal scaling

Automatic scaling without human interventions → Auto Scaling

→ There is no single solution that can be developed for all large scale systems, it varies use-case by usecase.

Maintainability

maintain existing software - fixing bugs etc {Oncall}

① Operability → making life easy for operations.

- ↳ Monitoring health of systems
- ↳ Debugging system issues
- ↳ software updates
- ↳ system Automation
- ↳ Oncall Rubooks and System dashboards ....

② Simplicity → Managing complexity.

- ↳ How different components are coupled?
- ↳ hacks introduced in system as short term solve.
- ↳ Complex systems makes testing a cumbersome task.
  - ↳ possibility of introducing a bug.

How can you achieve it?

↳ Abstraction

example → Java abstracts out memory clean up logic.

③ Evolvability → making changes easy.

system requirements change and it will requires modification in existing system, how easy is this process?

Subscribe YouTube channel for more such content.

Channel - MSDeep Singh

Happy Learning