Design Online Offline Indicator

Requirements:

user 1 -
user 2 -
user 3 -

Storage:

user -> online/ offline

int : bool

Access : Key - value

DB

Interfacing API :

GET /status/users? ids = u1, u2, u3, u4

GET /status/user? id= u1

Actor

API Server

Database

Updating the database :

POST / heartbeat :
The authenticated user will be marked as alive

Actor

API Server

Database

Every user Periodically sends "heartbeat" to the service. : 10 seconds
If we do not recive heartbeat long enough --> 30 seconds. : Offline

Storage :

User-id, last_hb (timestamp)
int, long

DB

If No entry exists - offline
If entry and entry.last_hb < Now() - 30 seconds : offline
else : online

How to approach system design

- Decide the core and build system arround it.
- core is use case specific and it could be database, communication protocol, Must have components - CDN, Cache

Another good approach :

- start with day 0 architecture
- see how each component would behave at next scale
- mitigate and re architect
- repeat

Points to remember :

- Understand the core property & access pattern
- affinity towards a tech comes second
- build an intuitation towards building systems

Scale Estimation :
1 user = 1 entry

1 entry : user_id, last_hb
4 Byte, 4 Byte = 8 Byte

1 M user = 8 MB
1 B users = 8 GB

Total Storage required = 8 GB

one DB Instance can definately hold the data

But, it can not handle the load ..

Therefore we need to partition the data.
each node handle a subset of the requests

Database
Database
Database

Sharded db

Can we do better on storage?

Requirement : if user is online or offline

we dont need to store all the user --> absense : offline

Let's expire the entries after 30 seconds.

Total entries = Active users

How to auto delete?

Approach 1: write a cron job

Approach 2: Offload this to datastore (preferred)

1st -
Not a robust solution, need to handle edge cases

if 1 B total users and 100K active then total entries = 100L

`Size of DB = 800 KB`

`Never re-invent the wheel !`

DB with KV + expiration --> Redis , DynamoDB

Redis Vs DynamoDB

O(1) write in Memory
Cost effective
Multiple env setup is easy

Maintanance
Vendor Lock In

Redis is preferred
Also, we do not need persitance

flow: post: Upon recieving th heartbeat : update entry in redis / dynamoDB with ttl = 30 second
get: check if entry exists?

How is our DB Doing?

Heartbeat is sent every 10 seconds.
So, one user in 1 minute sends 6 heartbeats.
if there are 1 M active users, our system will get 6M req/Min
Each heartbeat requests results in 1 DB call.

Our DB needs to handle 6M updates per minute

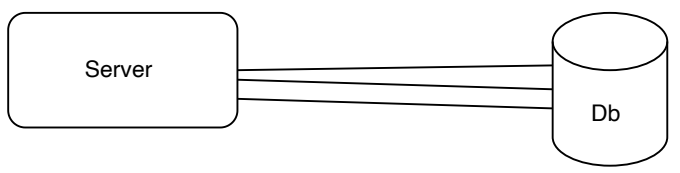
Note: In real world, Webscockets are used in such system.

How to make it better?

Async flow Not an option
What is hectic? Creating a new connection everytime

Connection Pool

Connection Pool



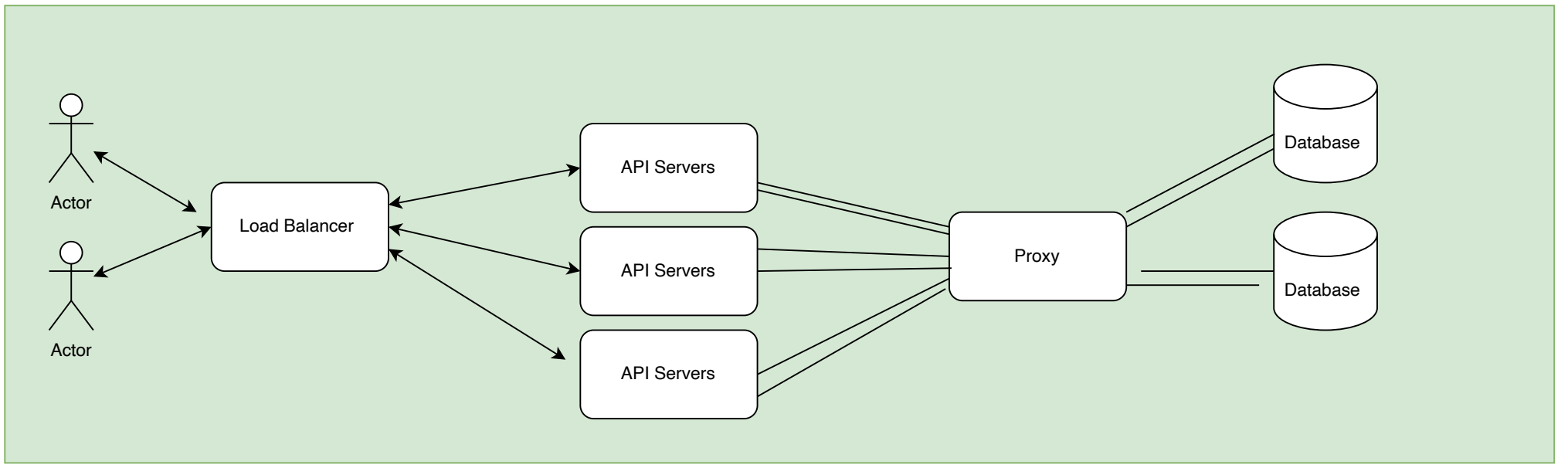
avoid 3 way handshake
save memory allocation in creating connection

New Product Requirement:

We show user: Active 12 mins ago
We cannot use auto expire approach, instead we would use have to store all data for all users

Similar System :

Failure Detection in a distributed system



Design a multi user Blogging platform

Requirements -

One user have multiple blogs
Multiple users

Database

Relational DB

Users

Id
Name
Bio

Blogs

id
author_id
title
is_deleted
published_at
body

Importance of is_deleted :
Soft delete

when user invoke delete, instead of delete we update

Key reasons: Recoverability, archival, audit
+ easy on database engine - no tree re-balncing

column type
body vs bio

long text vs short text
LONGTEXT VARCHAR

long text: stored as a reference ...
short text: stored along with other columns

Storing datetime in DB

datetime as datetime
- 02-04-2022 T 09:01:242

- Serialized in some format
- convient
- sub-optimal
- heavy on size and index

datetime as epoch integer
- efficient
- optimal,
- lightweight,
- cons: human readabilty from db

seconds since 1st Jan 1970

datetime as custom format (int)
YYYYMMDD - 20220402

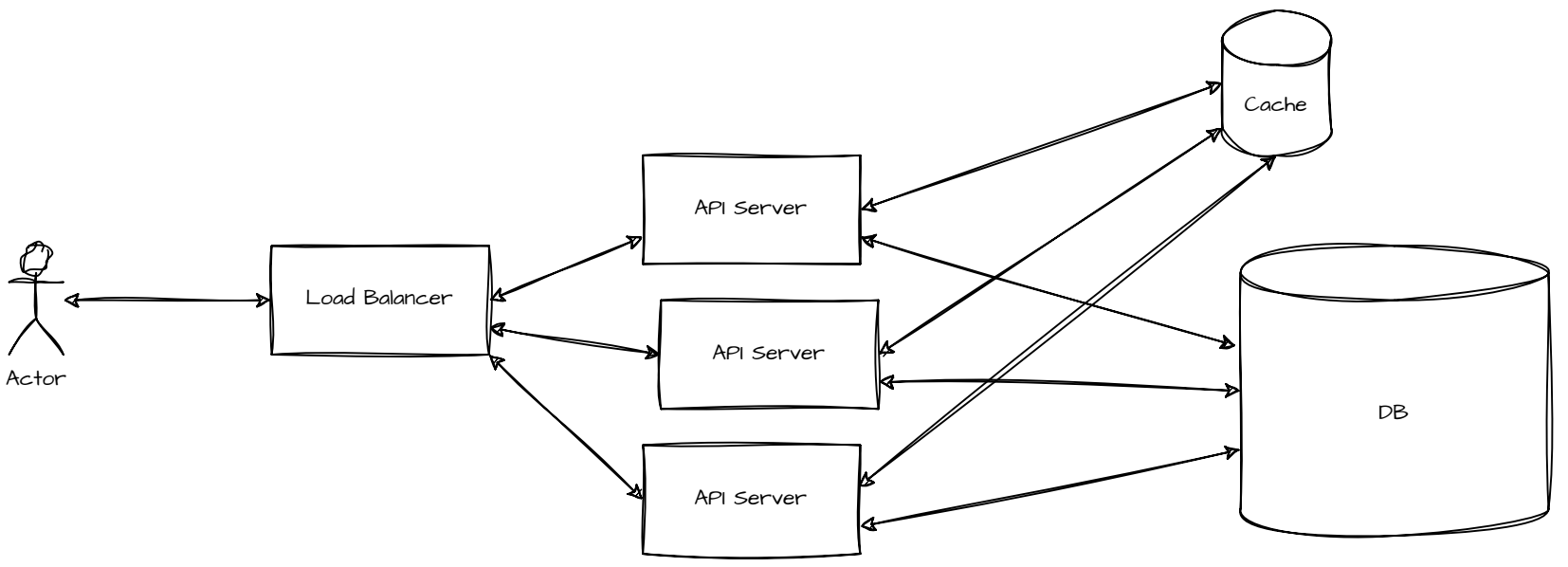
Caching

-reduce response times by saving, any heavy comuptation
* cache are not only RAM based .

* Typical use: redcue disk I/O or network I/O
Cache are just glorafied Hash Table (with advance datastructure)

Evaluate access Pattern & exploit.
Evaluate on static-ness of the data

Debugging is tough.
Need a Good Invalid strategy



Places To Cache:
Two level caching - Server RAM & Centeralized cache (redis) --> DB

Cache at API Gateway
CDN for Backend API Gateway
Nginex caching
Disk as a cache

Scaling

Ability to handle large number of concurrent requests

Two Scaling strategies

Vertical Scaling

Horizontal Scaling

Good Scaling Plan [In General]

Easy to manage
Risk of downtime
Hulk

Make infra bulky, add more CPU, RAM, DISK

Minions
Complex Architecture
Network Partitioning
Linear Amplification
fault tolerance

scale vertically -> then Horizontally

No Pre-Mature Optimizations

Concurrency

To Get faster execution -> Threads, Multiprocessing

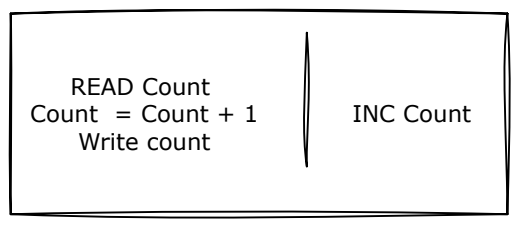
Issues with concurrency
- Communication between Threads
- Concurrent use of shared resources - db / in-memory varaibles

Handling concurrency
- Locks (optimistic & pessimistics)
- Mutexes and semaphores
- Go lock-free

Concurrency in our blogging platform

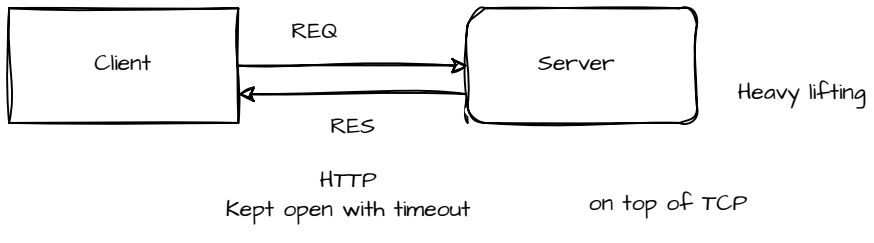
Two users clap the same blog, count should be +=2

We protect our data through -> Transactions
-> Atomic Instructions

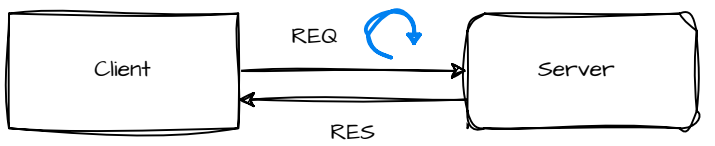


Communication

The usual communication



Short Polling



Eg: Refreshing cricket score
Checking if servcer is ready

Disadvantages :
HTTP Overhead
request and responses

Long Polling

Server response only when data is available

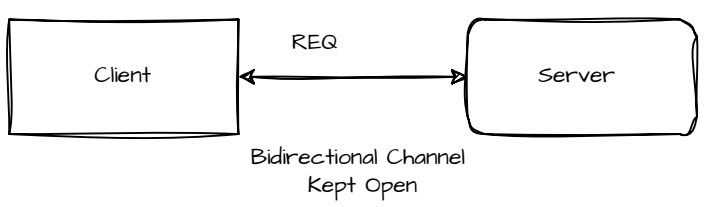
HTTP / (with timeout)

Short polling vs long polling
- short polloing sends response right away.
long polling sends response only when done.
connection kept open for the entire duration.

EG: EC2 Provisioning

short polling : gets status evert few seconds
long polling : gets response when server is ready

Web Sockets



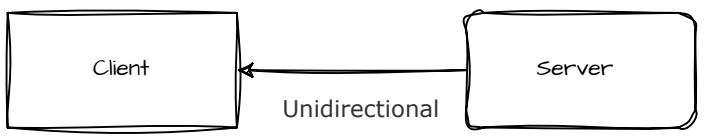
HTTP Based

Server can proactively send data to the client.

Advantages: realtime data transfer
low communication overhead

Applications : realtime communication
stock market ticker
llive experience
multii player games

Server side events



Http Based

proactively sends data to client

Applications : stock market ticker,
deployment logs streaming