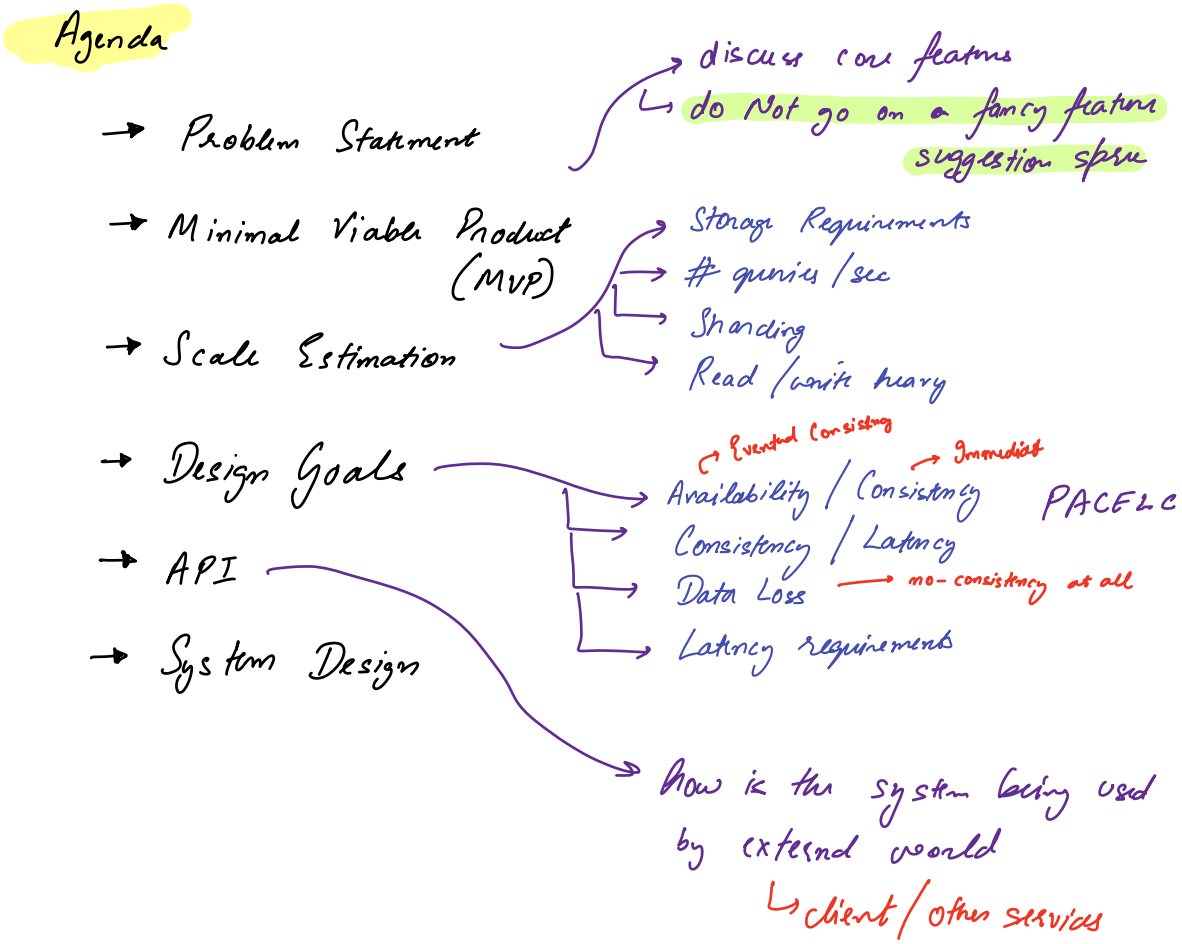

Agenda
Problem Statement
discuss core features
do Not go on a fancy feature suggestion spree
Minimal Viable Product (MVP)
Scale Estimation
Storage Requirements
queries / sec
Sharding
Read / write heavy
Design Goals
Eventual Consistency
Immediate
Availability / Consistency
PACELC
Consistency / Latency
Data Loss
no-consistency at all
Latency requirements
API
How is the system being used by external world
client / other services
System Design

Problem Statement

Facebook Messenger / whatsapp / Telegram / Discord / Slack / Teams

whatsapp → 256 people in a group max

Slack → groups/channels → 100,000 users in a group

Minimal Viable Product (MVP → v0)

- User registration / login → separate service
- Sender can send message to reciver
- attachments (Image / audio / video ..) v1
- online / offline v1
- read recipts / sent recipts v1
- group conversations v2
- recent conversations
- message history
  - ↳ all messages should be permanently stored on server side
- real time chat
- end-to-end encryption v3
- notification → separate service (v1)

→ discuss during interview

→ during class ∵ learning.

Scale Estimation

QPS — estimate yourself / ask interviewer

~~7B people in world~~ → ~~2/3 have access to net~~ → ~~5 bil. or 7 bil~~

~2B MAU at facebook → 50% are DAU → 1B DAU

each user sends 10 msg/day on average → 10 B message/day

Story 4 million messages on 1st day → Hackathon Messenger service

On day 1 we expect to get ~10 billion messages/day

facebook deals with > 70 B messages/day.

QPS
queries/sec

$$\frac{70B}{86{,}400} \approx 1 \frac{7 \times 10^{10}}{8.6 \times 10^{4}} = 10^{6} \text{ m/s} \Rightarrow 1\text{M msg/sec}$$

Storage requirements

Size of 1 message (on avg)

:+1:
:smile:
<3

Content ≈ 100 letters (100 B)

timestamp = 8 B
receiver_id = 8 B
sender_id = 8 B
attachment_path ≈ 50 B → large files/blobs → never stored in DB (S3)

Total ≈ 200B (size of each message)

70 B messages / day

Daily Storage requirement ⇒ 200 bytes * 70B

= 14,000 B bytes

≈ 14 TB / day

20 years of data = 20 * 365 * 14 TB

≈ 100,000 TB

≈ 100 PB of data in total

Sharding

Oh yes!!

Read / write heavy

→ every message is read

Read heavy

Write heavy

} Both read & write heavy

↳ every message sent → stored

Read heavy → caching

Write heavy → Sampled data

∵ consistency was not important

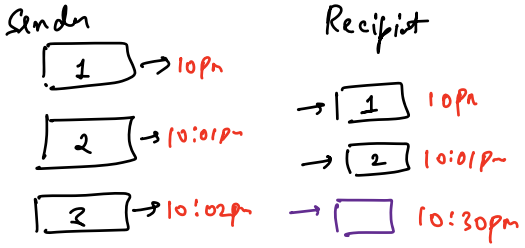
Design Goals

Availability vs Consistency

→ Communication for billion of human civilization

(Eventual Consistency)

(immediate consistency)



messages can get delayed

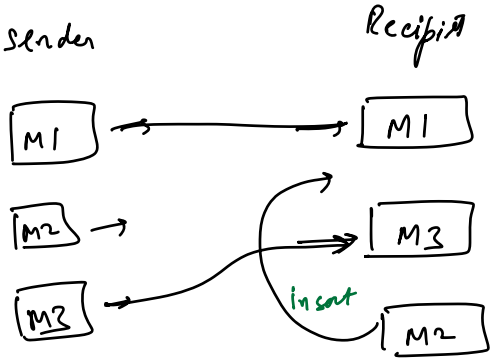
Sender sees that M2 was successfully sent at 10:02 but in reality it is sent at 10:30

✓

if a message is shown as sent → then it must actually be sent

---

Designing Data Intensive Applications → Martin Kleppmann



messages received out of order

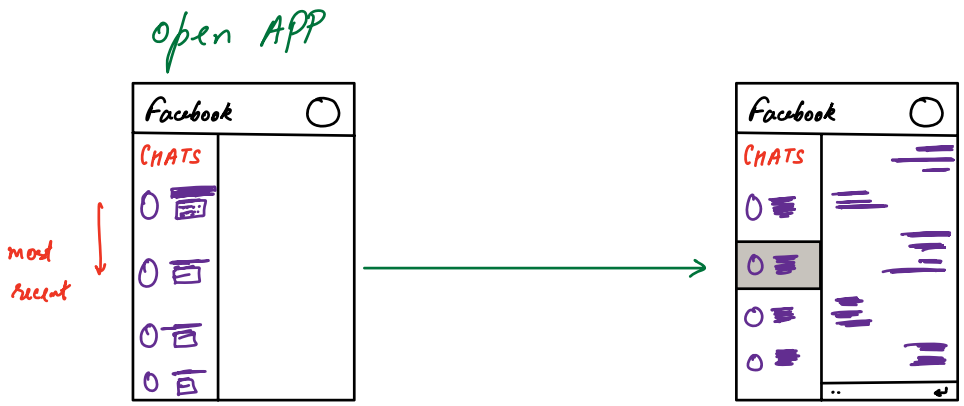
Consistency vs Latency → Consistency

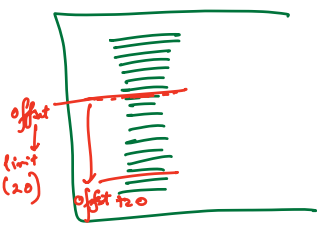
Data Loss → No!

Latency → pretty low

typeahead < 50ms
{ 100 x
messenger < 5 sec

API → how will external world (client / other services) use our system



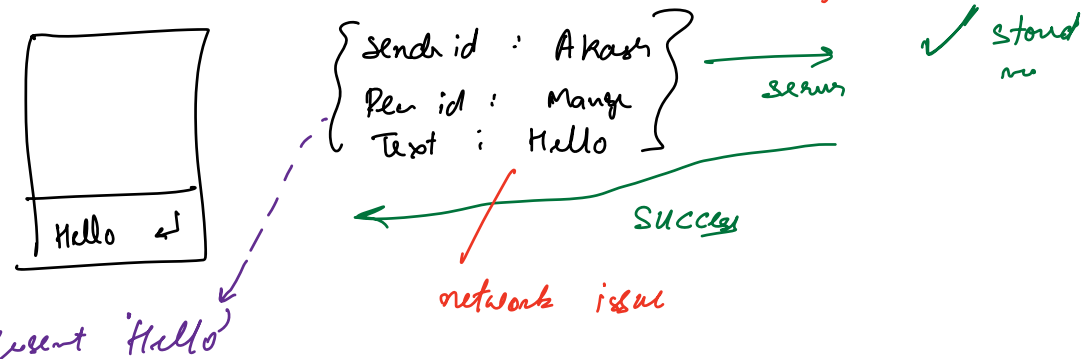


→ getConversations (user_id, offset, limit)
sorted by timestamp
pagination

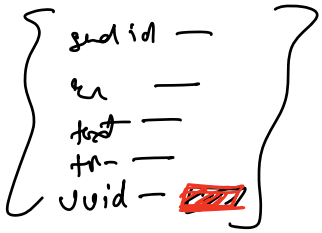
- getMessages (conversation_id / (user_id, friend_id) / (group_id) , offset, limit)

→ sendMessage (conversation_id / (user_id, friend_id) / (group_id), message, uuid)
uuid → unique to message

Idempotence → $f(x) = f(f(x))$ ⇒ idempotent function



We want to prevent the same message from being stored twice, but still allow multiple messages with same text if user wants it



10:34 → 10:40

System Design

Sharding

user_id → only work for 1-1 conversations

→ all conversations of 1 user must be present on 1 shard

any req for user will only read from 1 shard

get Conversations (user id, offset, limit) → easy → ∵ all the conversation if this user are in 1 shard

get Messages (conversation-id, offset, limit) → easy ∵ all conversation for this user
- ↳ friend
- ↳ group

↳ (sender, reciever)
↳ (group-id)

Send Message (conversation-id, message, uuid)
- ↳ (sender, reciever) → store in both sender shard & reciever shard. (consistency)
- ↳ (group-id) → fan out write
  - |group| = 10,000 users → 10,000 entry
  - super expensive ✗

idea → each group behaves like a user.

UI → send G1 → another in UI shard & G1 shard

read?

conversation id → (sender, receiver) 1:1
conversation id → (group-id) groups

getConversations (user id, offset, limit) → difficult
∵ to get conversation of a user we have to hit diff shards

Alternate have another separate db that stores
user → conversations data → conversation id, latest message, timestamp, unread count
sorted (most recent)
sharded on user-id

getMessages (conversation-id, offset, limit) → easy
↳ (sender, receiver)
↳ (group-id)
just hit that shard & get messages from the conv

SendMessage (conversation-id, message, uuid) → simple - just write to the conversation shard
↳ (sender, receiver)
↳ (group-id)

↓

however if an new

secondary db stores (user id → conversation (sorted))

when a message is sent to conversation → go to secondary db.

for each user that is part of the conversation, make the conv as most recent.

✗ expensive → apps like Slack → don't sort most recent conversation.

Slack still has secondary db to store user id → list of conversation ↳ not sorted by recency

Send Message Consistency

Sharding by user-id → send message → commit → sender should

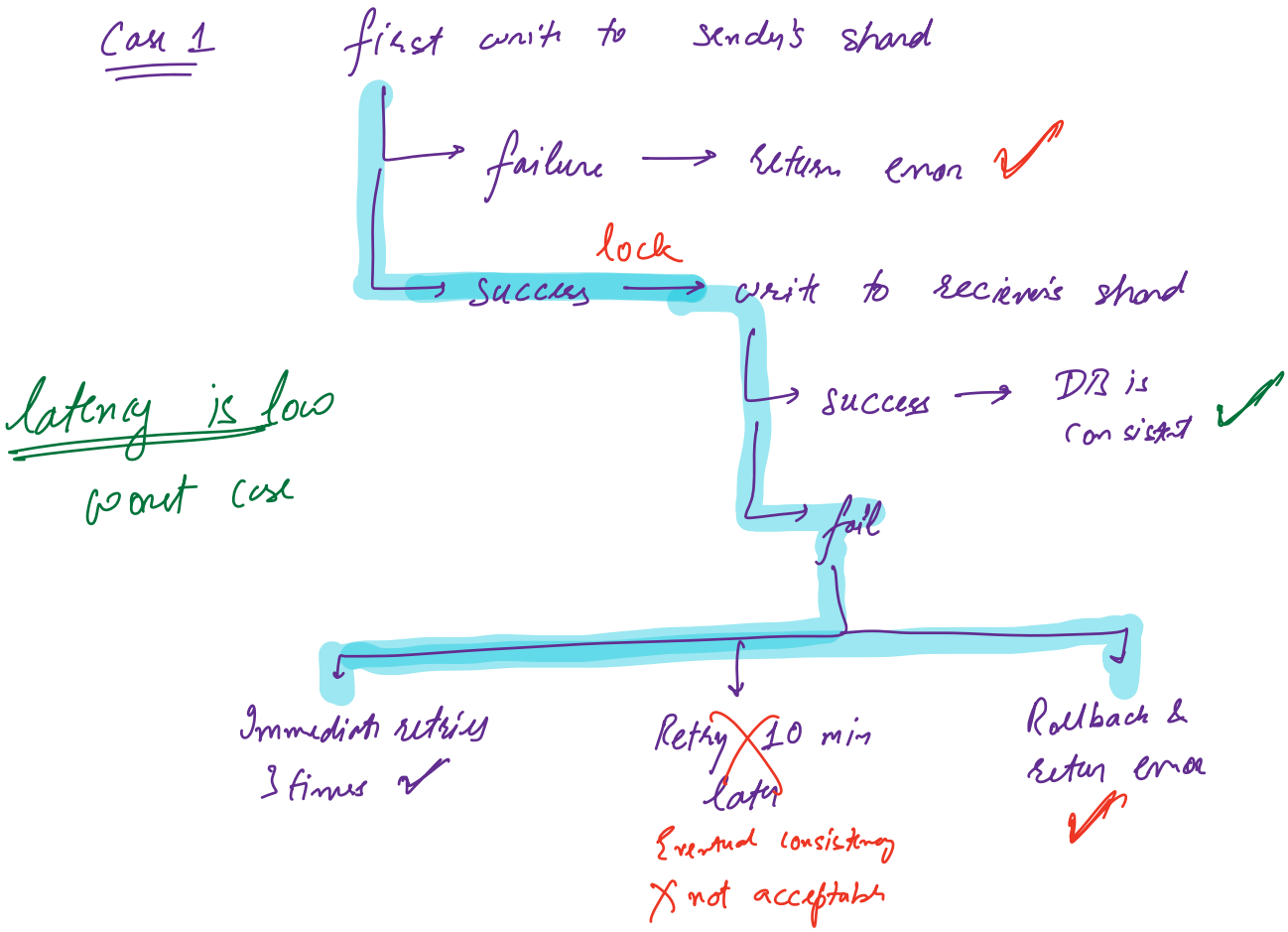
commit → receiver should

Problem → what to do if 1 write succeeds but other fails

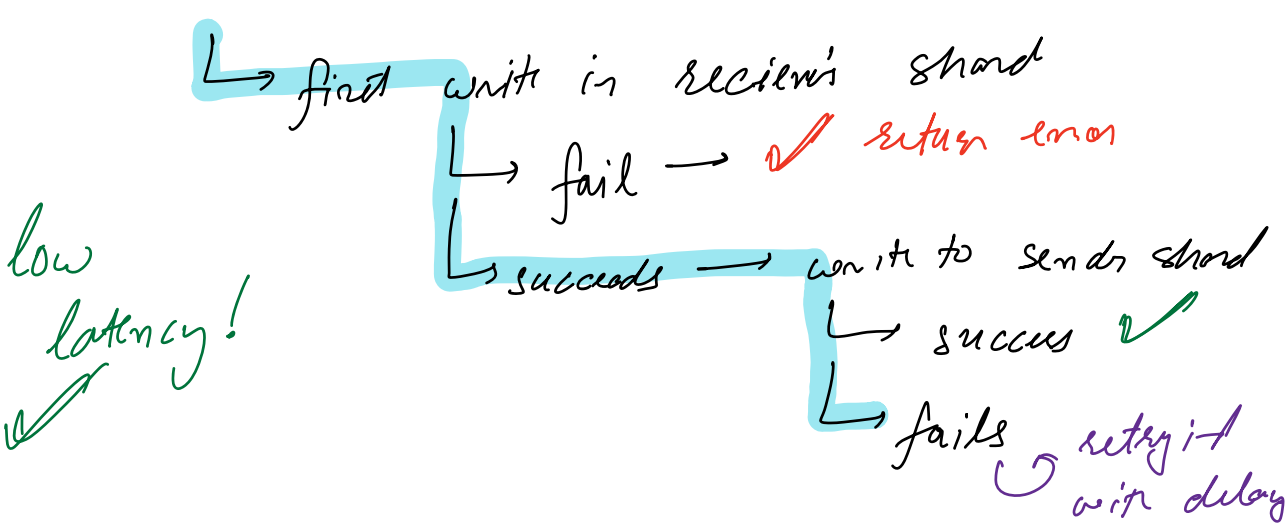
Sender → receiver

sent ✗ not seen

Case 1


first write to sender's shard
failure → return error
lock
success → write to receiver's shard
success → DB is consistent
fail
Immediate retries 3 times
Retry 10 min later
Eventual consistency
not acceptable
Rollback & return error
latency is low
worst case

Case 2


Sender → Receiver
first write in receiver's shard
fail → return error
succeeds → write to sender shard
success
fails
retry it with delay
low latency!

note → when sender refreshes app (browser)
then sender will not see their sent
message
↳ still get info from app cache ✓

Choice of database → both read & write heavy.
↳ requires a lot of sharding.
↳ No SQL

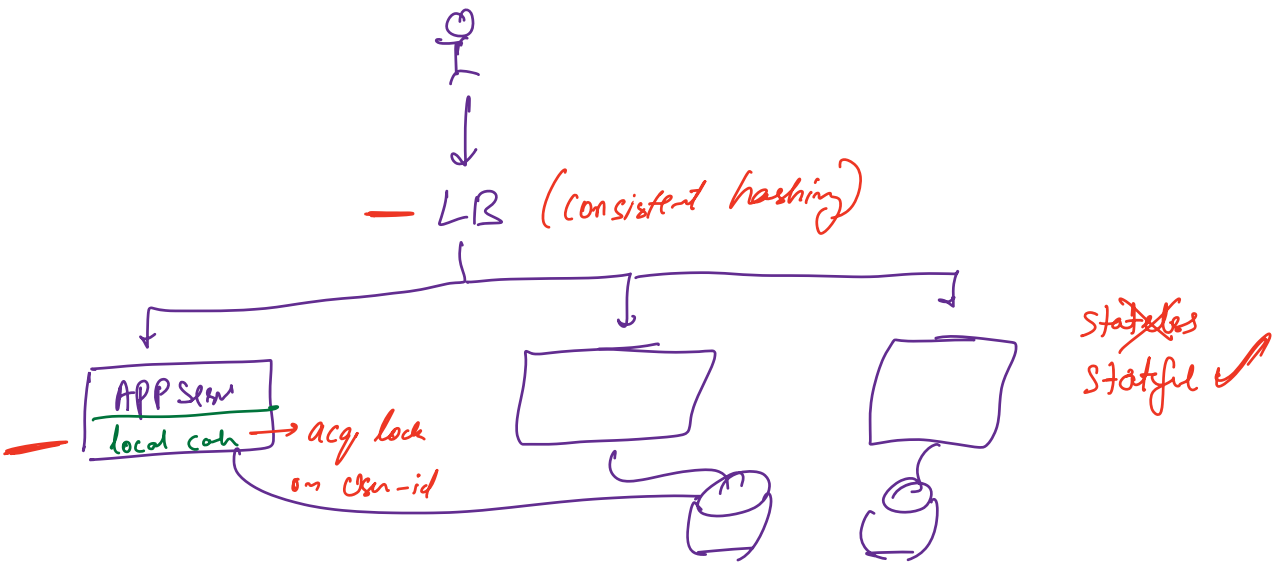
No DB in the world optimized for both reads & write

Commit to either read heavy or write heavy.
↓
needed ∴ can read immediate consistency

↳ Ensure that most reads never go to db
↳ heavy caching. → Sharded
↳ Cache is consistent with DB
↳ write-through cache

↳ make sure that we handle high concurrency in cache.

Don't use global cache → local caching.



Pros → ① Scale horizontally

② Race condition on handled.

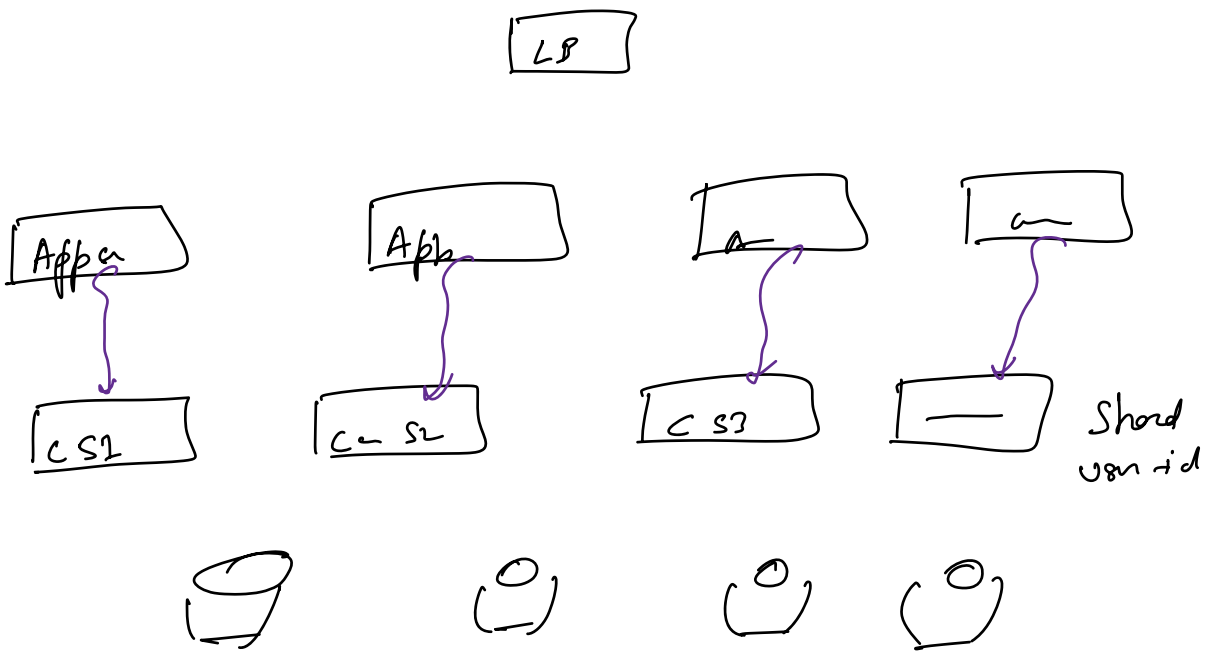
Cons → ① If app server goes down. → Until user is assigned a new server

↳ cold start of cache for new app server.

Service will be unavailable to that users

Column family → : Recent messages / conversation

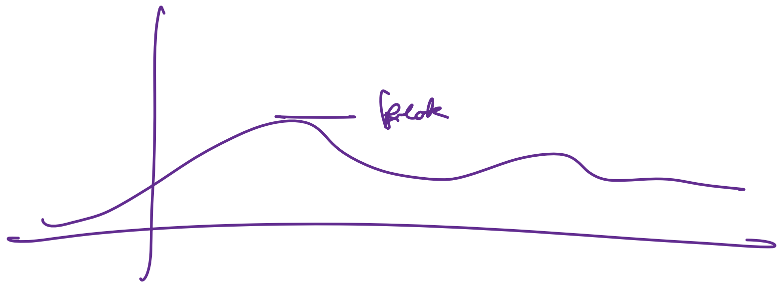
Write optimized → Hbase



1 Million messages / sec (avg traffic)
→ Write → 1
→ Read → 3

1 appsn can handle → 1000 writes / sec (If each write is for a diff user)
→ 1000 reads / sec

↳ 1000 app servers.



Websockets → persistent, two way connection
HTTP/HTTPS

1:1 conversation → userid sharding

group convrsatn → Conversation-id sharding
→ secondary DB to store
user → [conversation list] data
→ small group
sorted by recency or not.

Sharded on user-id.

Scale HLD batch → 1000 users.
20 → group-id

Vishal → Scale HLD Batch
msg
user id = 300

20
Vishal msg

Aman ← Server
send

R+W vs X
↑ ↑

Cache ⟶ 10 year old messages ✗

↳ last 10 days of data

70B msg/day *200B *10

= 140 x 10^9 Bytes

= 140 TB data in cache

Idempotence

for all x $f(x) = f(f(x))$

if you apply the function multiple times the result doesn't change.

≠ fixed point means Some x

abs(x)

abs(-10) = 10

abs(abs(-10)) = 10

floor(x)
ceild(x)

$\frac{d}{dx}$ X

$\frac{d}{dx} e^x$

API Impotent

GET (read)

GET ashrith/message/ ← msg

GET ashrith/message/ ← msg

GET ashrith/message/

POST → insert
↳ usually not idempotent

POST ashrith/notes/

Payload { title : 'H2D'
cat : '___'
~~ug uid~~ }

id = 10 [inserted to db
id = 11 [another post is a
id = 12 [

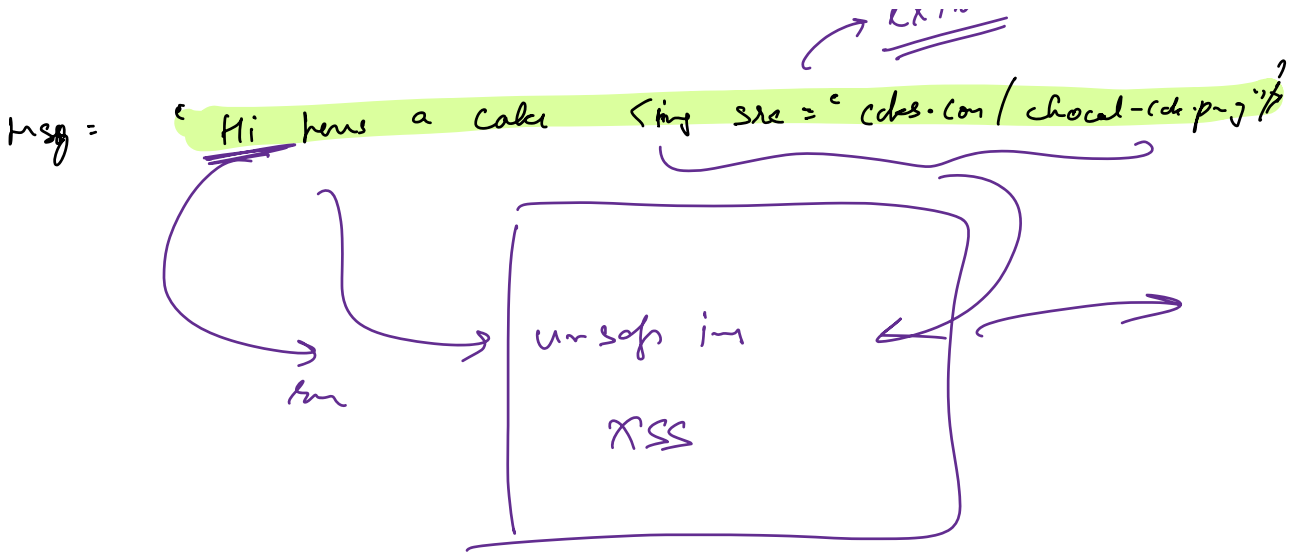
PUT → update
↳ always idempotent

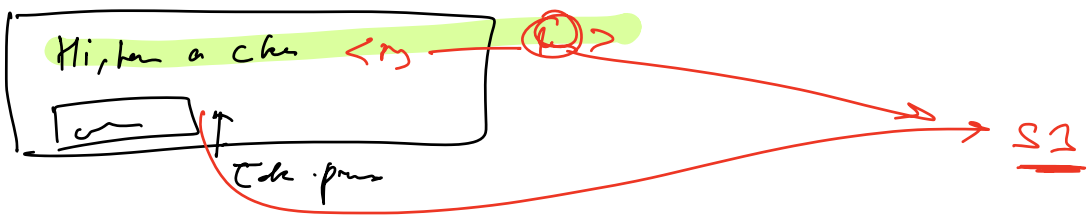
PUT ashrit/notes/10

Payload { title : 'H2D Scd'
cat ______ }

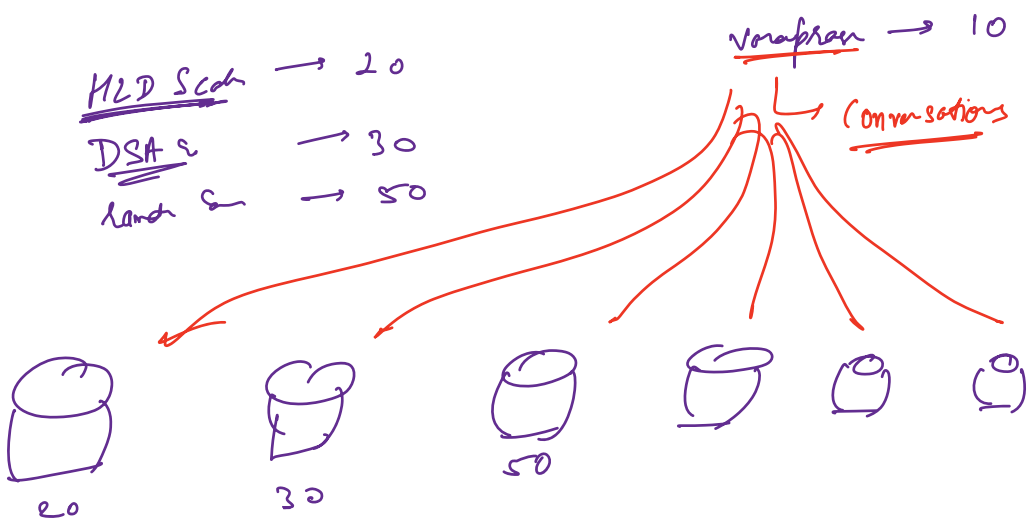
→ updated
→ mostly no change

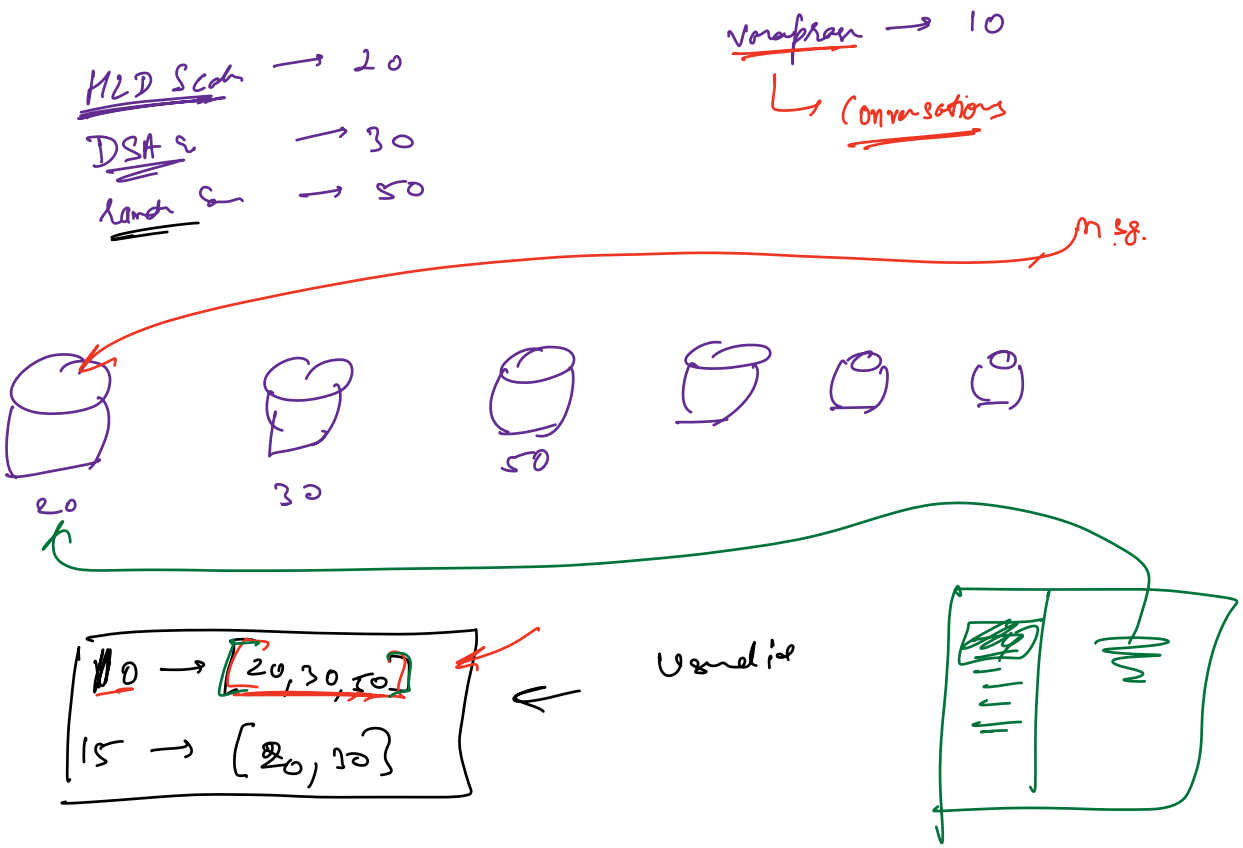
...ntamd.





Conversation-id [group id / (recv/snd)]





10^5 MAU

↳ On avg each user

makes 10 purchase each day

3 B MAU for 30 days

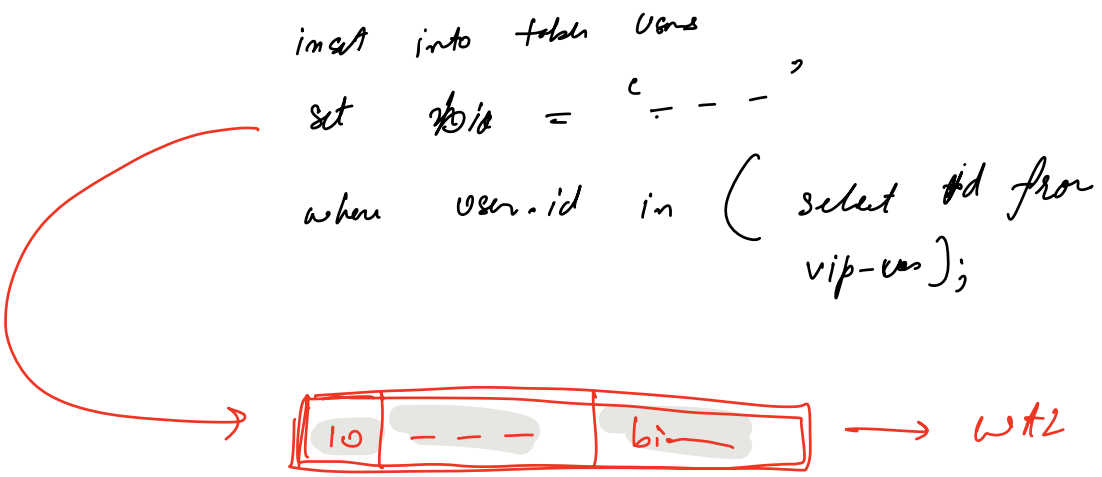
$3 \times 10^9 \longrightarrow 10^5$

10^4

10^6 purchase per day

$$\frac{10^6}{86,400} \approx \frac{10^6}{10^5} \approx 10 \text{ req/sec}$$

WAL



SSTable → disk

Sparse Index → memory.