Design Facebook Messenger

Tuesday, March 19, 2024 8:51 PM

Functional Requirements
1. Sending / receiving message
2. Store a message
3. Show user status
4. Show chat history

Extended requirements
1. Should carry chat group chat
2. Push notifications

Non-Functional requirements
1. Low latency
2. Consistency
3. Reliability
4.

CAPACITY ESTIMATION

500M users — Stored for next 10 years
Each user sends 40 messages per day.

Traffic
500M × 40 message
= 2000 × 10^6
= 20 Billion messages/day.

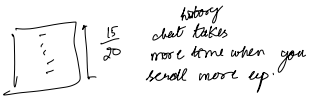
KB - 10^3
MB - 10^6
GB - 10^9
TB - 10^{12}
PB - 10^{15}

Storage
$20 \times 10^9 \times 100$ bytes $\Rightarrow 2 \times 10^{12} = 2$ TB/day
2 TB × 365 days/year × 10 years $\Rightarrow$ 7300 TB
7.3 PB

Bandwidth: → will be in seconds
2 TB per day / 24 × 60 × 60

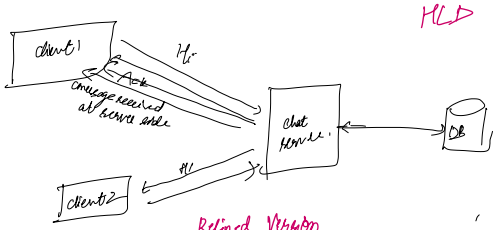
claimed as both incoming and outgoing
(because 2 users will chat simultaneously)

Cache: 2 TB × 0.2 $\Rightarrow$ 0.4 TB

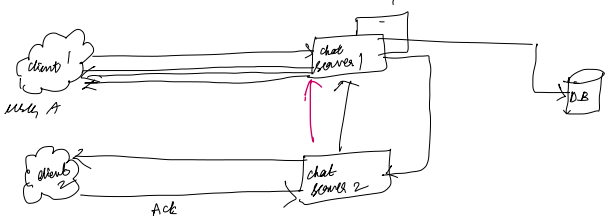


history chat takes more time when you scroll more up.

HLD



Refined Version



Hashtable
userId : connectionObject

What if the user B is offline?

Push → long polling → specific endpoint for the messages to be sent out
→ web sockets — persistent connection, 2 way communication (load to the server increases)

Pull → drawback, all the clients might ask and it increases the load on server

How to divide users between servers
500 × 10^6 / 50000 = 10K
↓
use hash(userId)

Push + Pull
(client online after sometime then pull)
(when data is available on server)

If chat server 1 fails we have a standby ready
Its hard to send connections to backup server
send all the connections to new backup — user should resend the message.

We need chunker if we edit the stuff

5:01 pm user A
5:02 pm user B
5:10 pm user C.

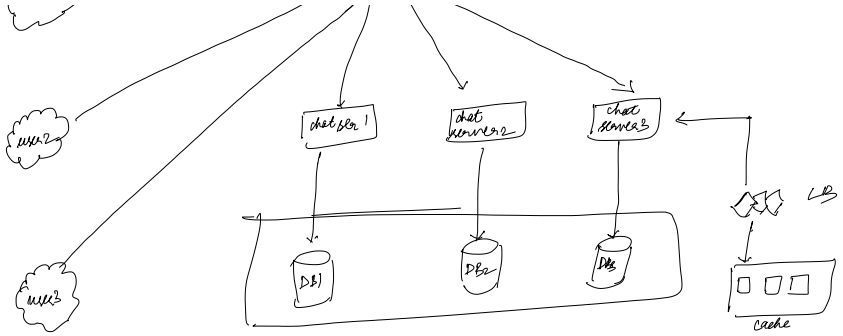
Time stamp
Queue

can't have diff copies
to store sequence number

HBase is a NOSQL database

For every user,
one store — messages, timestamp, sequence numbers — bundled into one object and store

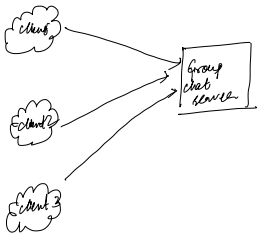
LOW LEVEL DESIGN



→ we use it for pagination

GROUP CHAT

storing
{groupId: [users]}



Cache Eviction: LRU

DB Partitioning -

hash(userId) → pin pointed to a server ✓ (this makes more sense)
hash(messageId) → can not be in single server.

Load balancing - hashtable will send out the requests

{ userId: connection objects }