

Facebook Messenger-Whatsapp

Problem Requirements

- 1) Support group chats with up to 10 users
- 2) Sending messages
- 3) Receive messages in real time
- 4) Persist messages so that users can access them from other devices (or load older conversations)

Capacity Estimates

- 1) 1 billion total users
- 2) Each user sends 100 messages a day
- 3) Each message is around 100 bytes (including metadata)
- 4) $1 \text{ billion} \times 100 \text{ msgs} \times 100 \text{ bytes} = \underline{10 \text{ TB per day}}$
→ ~4 PB per year

Chat Members Database



We should be able to fetch all chats for a given user

→ Recall left pane of initial diagram

→ Want all chatIDs for given user to be on same shard, next to each other and dis-

User Id	chatId
3	12
3	19
10	12
10	65

m
index + partition key

To avoid causal dependency write conflict of adding and removing a user from a chat, let's use single leader replication → set CHAT?

→ This DB likely not a performance bottleneck, let's just use MySQL

Users Database



Schema → User Id, email, password hash

Partition on User Id

While this table will have a lot of rows, it is also unlikely to be a bottleneck as it will not be written to or read from nearly as much as our messages table will

→ For simplicity's sake, let's stick to MySQL again

Messages Table



We want to make sure that getting all the messages for a given chat is fast!

Schema: chatId (partition key), timestamp (sort key), message, metadata

2	1000	"Hi!"	From: kate
2	1003	"Send Bobs"	From: Jordan

→ Messages for chat already on same partition and pre-sorted!

→ Timestamps ensure consistent ordering on all devices (order by TS on frontend)

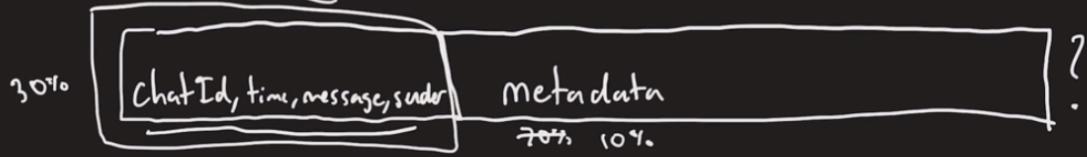
 → Distributed timestamps are unreliable, but does it matter?

Messages Database



What should we optimize for?

Reads → B-tree index, column oriented storage? (HBase)

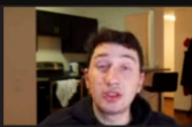


Writes → LSM tree index, leader less replication

↳ Cassandra?

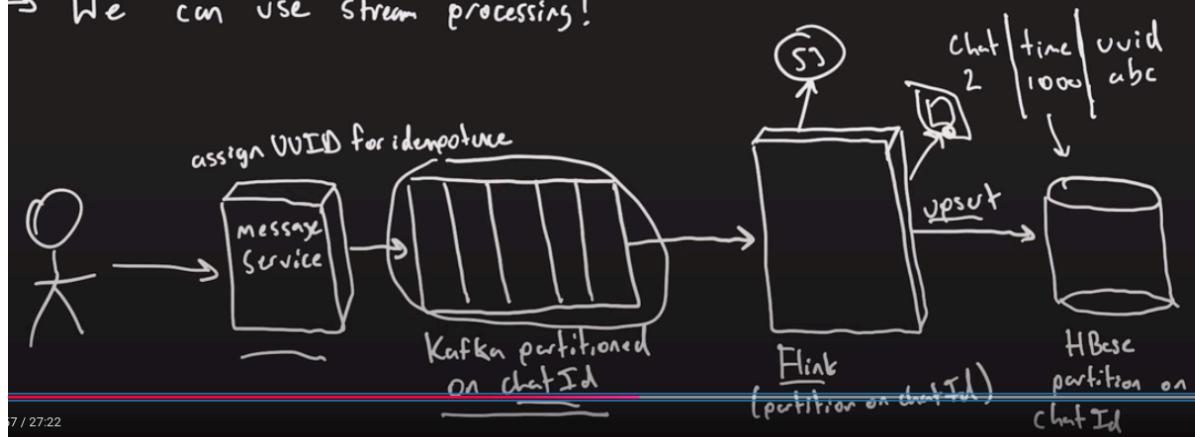
Maybe we can get the benefits of HBase without having to worry about write throughput!

Message Delivery



It would be great to be able to read messages from HBase while being able to quickly ingest them in our system!

→ We can use stream processing!



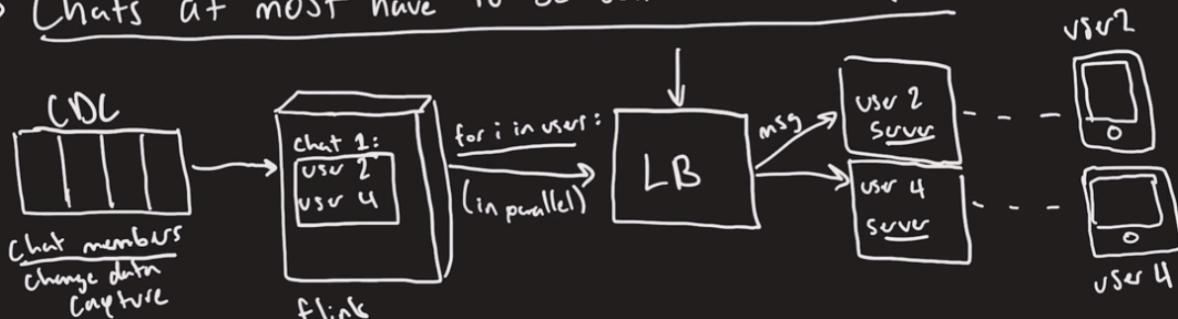
Message Routing



To minimize load on client devices we want them to maintain as few active connections as possible

→ Each user can just maintain a connection with one chat server

→ Chats at most have to be sent to 10 different places



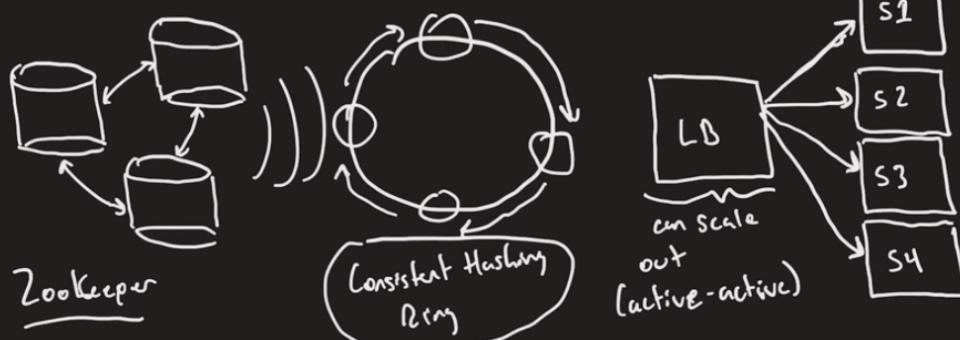
Connection to Chat Servers



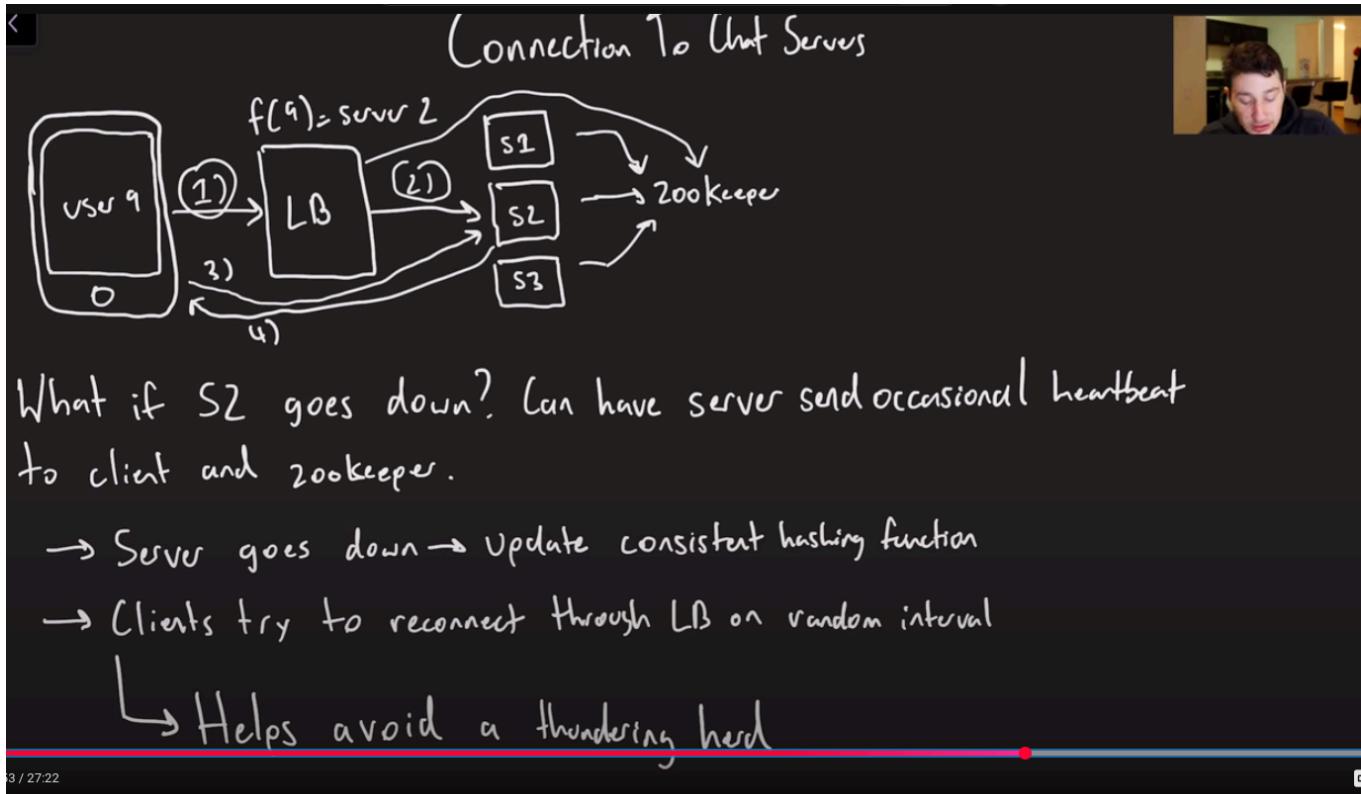
We need to reliably know the function to connect a given userId to a Server for:

$$f(userId)$$

→ Sending messages
→ Receiving messages } allows us to use BiDirectional realtime communication!



the load balancers will listen to zookeeper

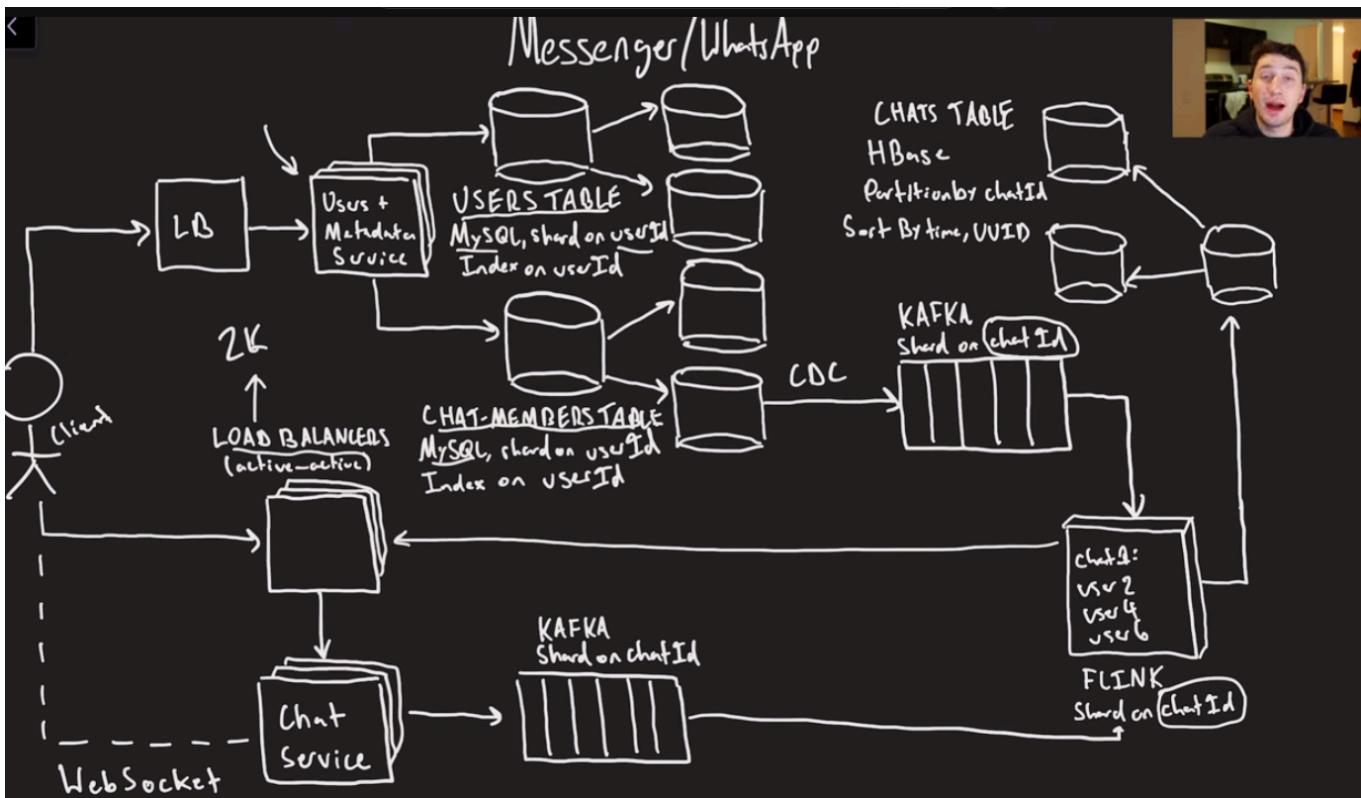


thundering herd problem-<https://medium.com/@venkteshsubramaniam/the-thundering-herd-distributed-systems-rate-limiting-9128d20e1f00>

<https://www.linkedin.com/pulse/stop-thundering-herd-problem-before-starts-mulualem-eshetu-16jfe/>

<https://encore.dev/blog/thundering-herd-problem#the-thundering-herd-problem>

-
- Realtime Connection Protocol
- 1) Sending and receiving lots of messages, don't want to resend headers every single time
 - Probably not long polling
 - 2) Want to have control over when clients reconnect to server
 - Maybe not server sent events (does this automatically)
 - 3) Want bi-directional communication
 - We should probably use WebSockets



If a user need to write a message it would send to the webSocket server which would contact kakfa which is sharded on chatId , kafka will contact flink which is also sharded on chatId and get all of the users in that chat,

then flink will again contact the loadbalancers and get the servers for it to connect to and send the messages to those servers .the servers will then look at its local webSocket connections and send messages.