5) SYSTEM DESIGN OF FACEBOOK MESSENGER

1) Requirements
(i) User can send text message to one user at a time
(ii) Online/Offline status of other users shown be shown
(iii) Persistant storage chat
(iv) Daily 500M active users and on average 40 messages are send with 100 bytes of size

Total Messages :
500M * 40 = 20B

Total Storage :
20B * 100 bytes = 2TB

For 5 years , 5 * 365(days) * 2TB = 3.6 Peta Bytes

2) High Level Design :
Flow of Events happens as below

(a) User A (Sender) opens connection with chat server and sends message.
(b)Chat server receives the mesaage and sends acknowledgement.
(c)Chat Server stores the message in Database and then sends the message to User B (Receiver)
(d) User B recieves the message and sends acknowledgement to chat server.
(e) Chat server notifies User A that message has been delivered successfully.

3) Detailed Design

(a) Message handling (send / recieve) - we can use PUSH AND PULL model, but PULL model (LONG HTTP POLLING) will be efficient for us.

Chat server can maitain all the connections of active users using HAST TABLE with key as Userid and value as connection object.

As there will be 500M active concurrent connections, and modern day server can handle 50K , so we will need approx 10k chat servers.

We can add software load balacer which helps to store connection objects to different chat servers based on particular logic e.g (hash) of Userid % 1000

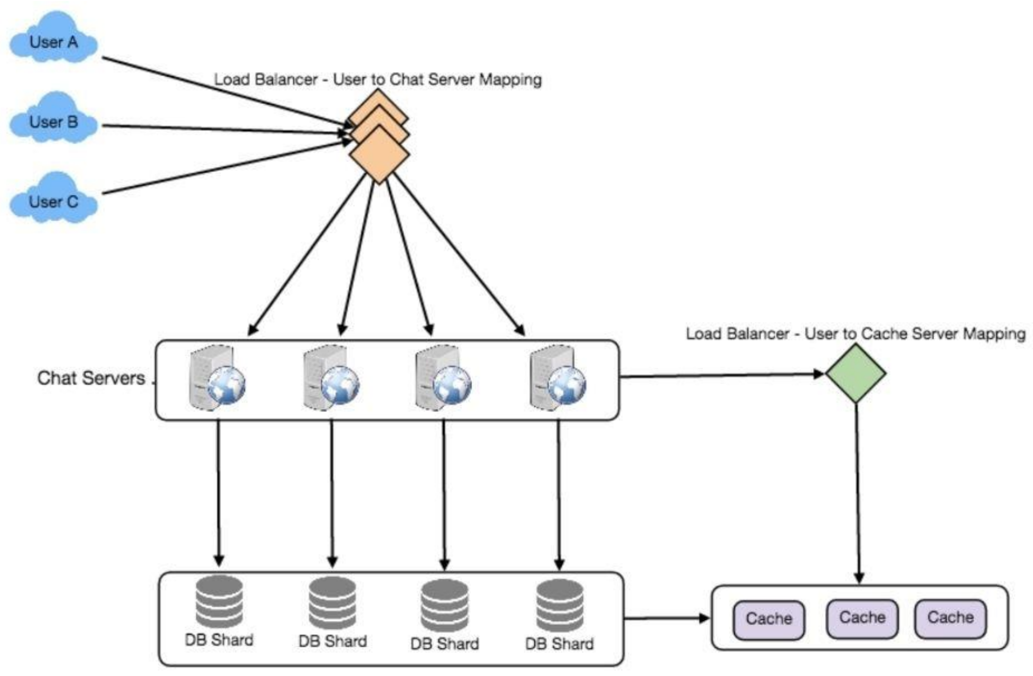
In order to keep sequence of messages appropriate, we can use SEQUENCE NUMBER associated with each message for each client.

(b) Storing and Receiving the message from Database - On receiving message by Chat server it can start a seprate thread to store data or send asynchronous request to db and at the same time send that message to reciever.

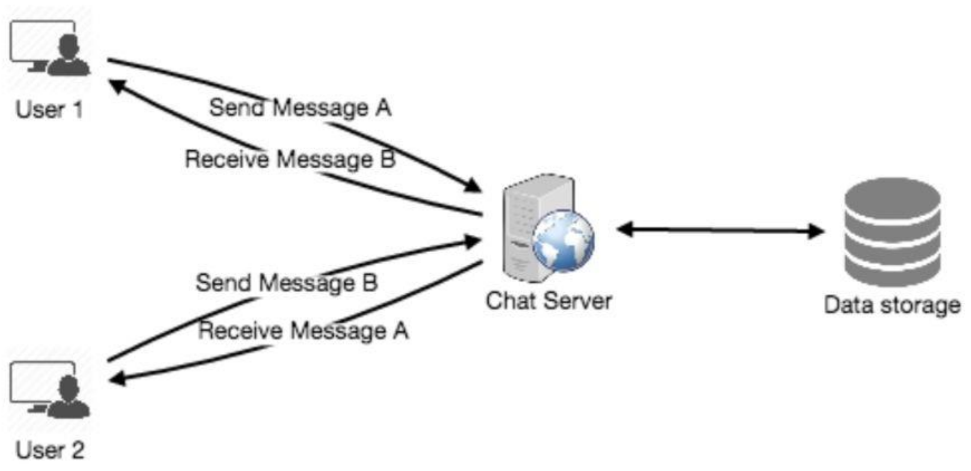
We can have NO-SQL database like wide column Db e.g Hbase as we need high rate small updates and fetch range of messages quickly.
Hbase is built on top of (HDFS) Hadoop distributed file system. It stores collection of new data in its memory buffer and once buffer is full it writes into disk.
This way write and reads are faster.
Also wide column Db helps to store data in mutiple columns.

(c) Managing user status - As we are storing active connection objects in our chat servers, it becomes easy to manage status.
We can brodcast status change event of user to relevant users.
Since 500M active users, it will be a heavy operation and consume alot of resources.
We can optimize it as below
(i) Pull status of friends list on app start.
(ii) If we send message and user is offline then we can update status change
(iii) If user is online, we can delay the broadcast as user might go offline quickly.
(iv) Whenever we chat with new user, we can pull status at that time

4) Data Sharding
We can shard data based on userId rather than messageId as it will quickly fetch being in one shard


User A
User B
User C
Load Balancer - User to Chat Server Mapping
Chat Servers
Load Balancer - User to Cache Server Mapping
DB Shard
DB Shard
DB Shard
DB Shard
Cache
Cache
Cache

Detailed component design for Facebook messenger


User 1
Send Message A
Receive Message B
Chat Server
Data storage
Send Message B
Receive Message A
User 2