Lecture 14 Week 8 Class 2 02 July, 2021

System Design, Performance Scaling & Load Balancing

We will be learning these topics through building a Tinder app with these features -

- → Store Profile
- → Recommend Matches
- → Note Down Matches
- → Direct Messaging

First e Monolithic Architecture hishbe start korbo. Then user base grow korle pore aste aste performance scaling korbo.

① Static Content Tier | Client side Languages - HTML, CSS, JS [redacted]

② Business Logic Tier | Server side programming - PHP, Node

③ Permanent Storage Tier | Data storage Engine - RDBMS, Distributed

Ei 3 ta tier ekta server e rakhle Monolithic. Business tier ke bhanglei sheta Microservice hoye jabe, ar Monolithic thakbe na.

Aste aste user base grow korar por dekhlam unresponsive behaviour start hoilo, e.g. image load hoite time lagtese / load hocche na.

Eta fix korte first step nibo -

Performance Tuning

- → Existing code base improve kora
- → ***How can we do that?***
  - ◆ Refactoring source code
  - ◆ Convert unnecessary Synchronous functions to Asynchronous functions
  - ◆ Logic improvement
  - ◆ Caching

Performance Tuning korar capability shesh hoye gele next step -

Vertical Scaling

- → Existing server kei spec up kora

Horizontal Scaling

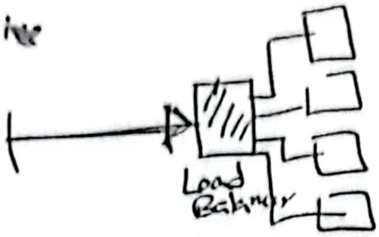
- → Number of server increase kora

Erporeo ek shomoy giye amar Business Logic Tier ar load nite partese na. Ekhoni Microservice Architecture implement korar kotha vabbo na, cz eta onek costly. Rather we will do at first -

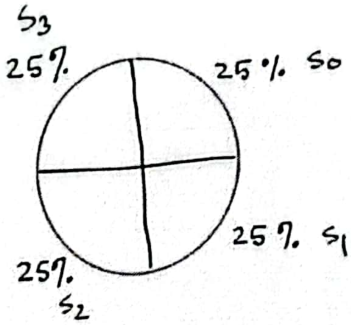
Load Balancing

Business tier ta ke divide kore 4 ta server korlam suppose.

- → Ekta user er data ekta particular server e cache kore rakhlam to get faster access. Shob gula server e sheta cache kore rakha is not a good design.



→ User request first e Load Balancer e jay. Shekhane Consistent Hashing er maddhome decide hobe kon server e sheta forward hobe, then oi particular server e forward kore dibe.



Business tier er 4 ta server e ekhon 25% kore load portese, jeta age purata ekta server er upor porto.

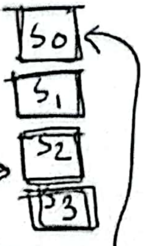
Consistent Hashing

Request ID → r_1

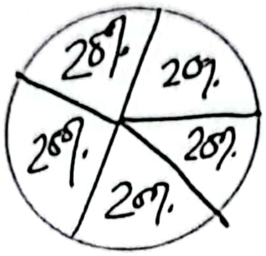
contains info about userID

'n' servers

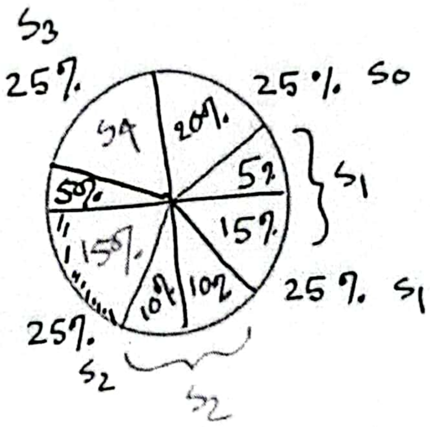
$$\text{hash}(r_i) \ \% \ n = \frac{0 \sim n-1}{0-3}$$



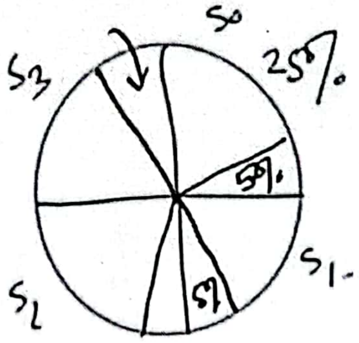
What if we add a 5th server?



Ager 25% gula ekhon 20% kore divide hoye jacche.



But this division has a problem. Ager 25% e jei cache data gula chilo, ei division er por shegula ar pabo na. Taile ekhon solution ki?



Rather we will divide like this. 4 ta server theke equal amount of space notun server ke dibo. Prottek 25% theke 5% 5% kore nibo arki. Although ekhaneo oi 5% user er cache data lose hocche, but it is better than losing all user's cache data like before.

Load balancing koreo temon labh hoilo na. Ekhon koi jabo? MicroService Architecture

Monolithic vs De-coupled Business Tier

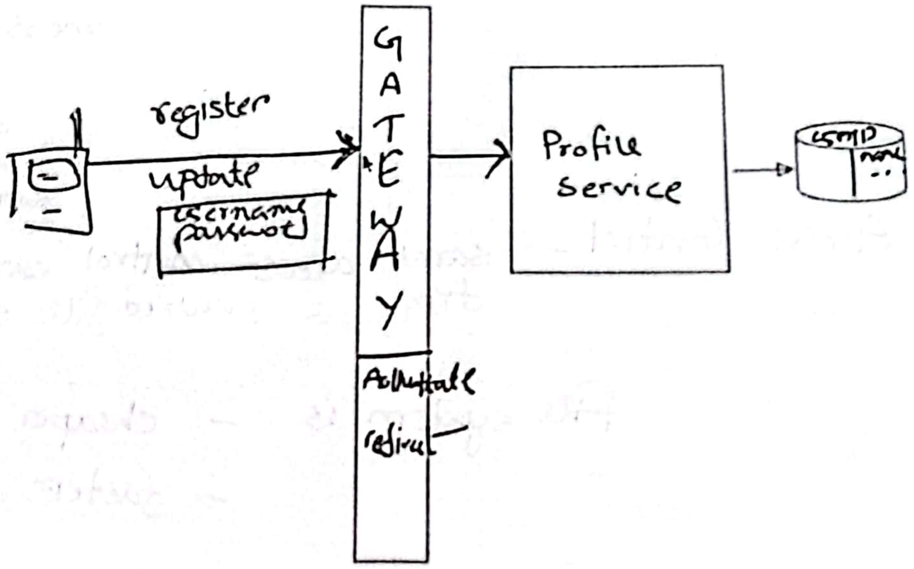
→ Usually services use separate Database/Tables

Tinder as a MicroService Architecture

1. Store Profiles (Images per profile - 5)
2. Recommend Matches (Active Users)
3. Note down Matches
4. Direct Messaging

Egula decouple kore koyekta service e vaag korlam.

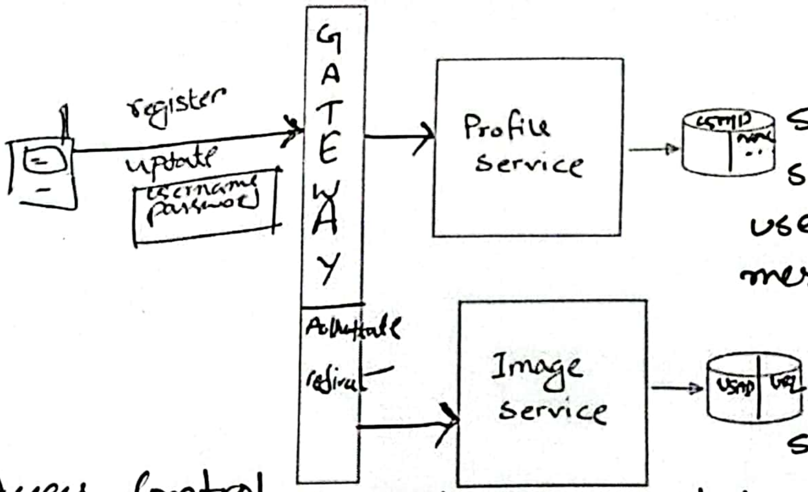
Let's store profiles first -



Now, store images -

→ **File vs Blob**

- **Mutability**
  - Ekta image change korle pura file / whole image change kori, mukh rekhe shudhu amar dehokhan change kori na. So file diyei hoye jacche, Blob dorkar nai.
- **Transaction**
  - Eta kono secured emon kisu na je eta fail hoile database inconsistent hoye jabe. So Blob er dorkar nai.
- **Indexes**
  - Tinder e ami image diye search dibo na google image er moto. So Blob er dorkar nai.



By using a separate gateway service, you can use different messaging protocols.

store images seperately from profile services for analytics purpose

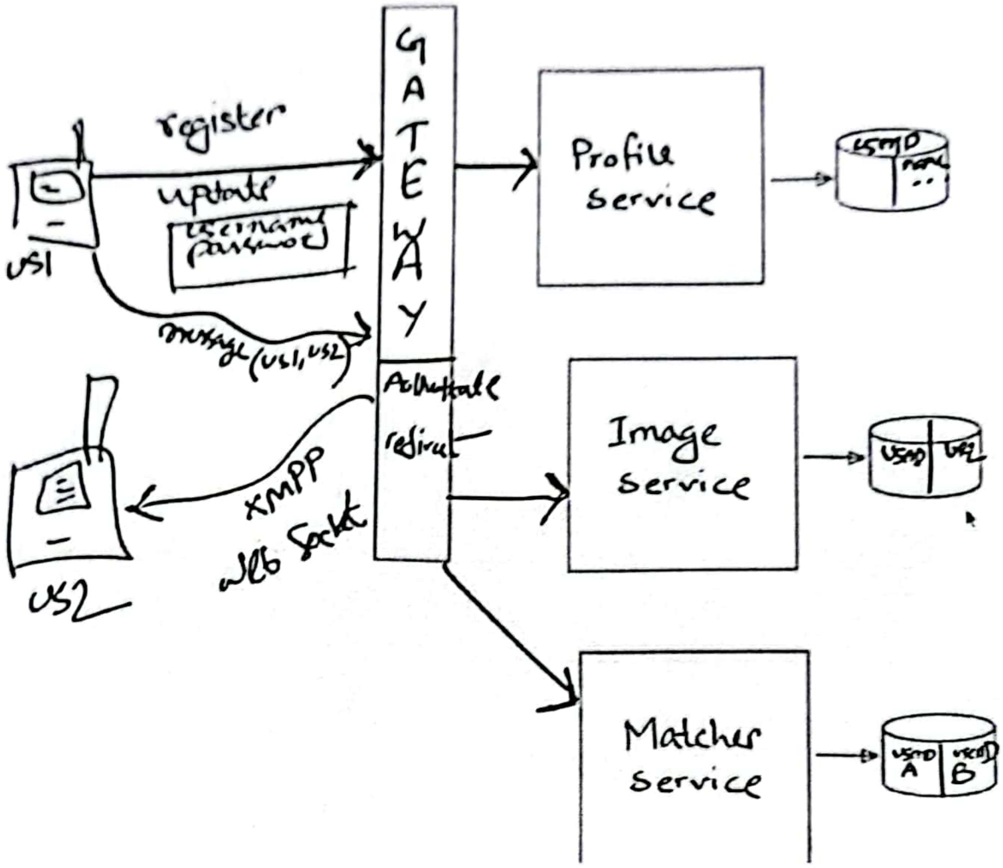
☆ Access Control — same access control can be obtained from a secured file system.

Store profiles Done!

File system is – cheaper
– faster
– You can build a CDN on this.

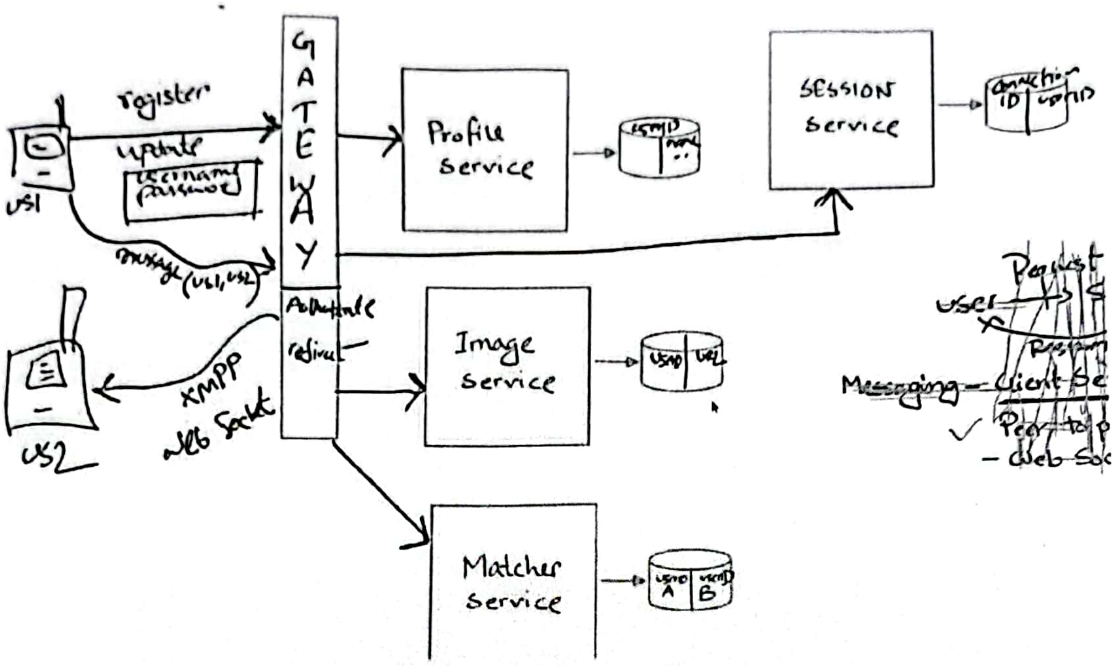
In DB you have to store –
profileID, imageID, FileURL.

case, we need something peer to peer (e.g. XMPP), je keu je karo sathe communicate korte pare.

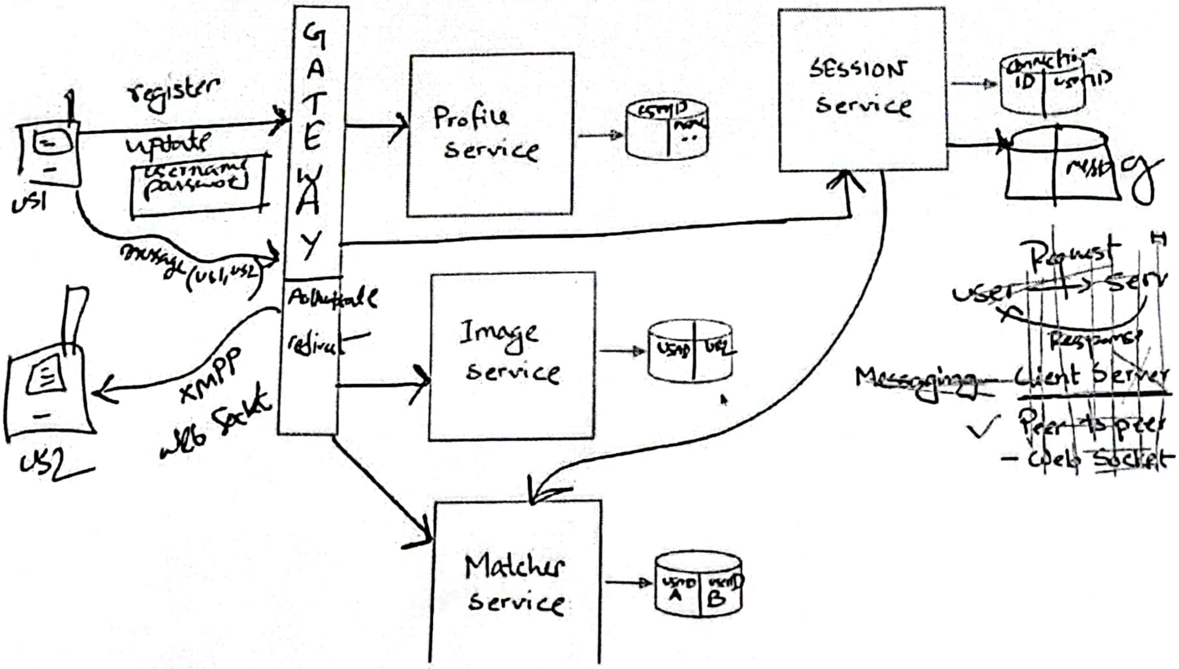


Tinder e to onek user ase. Jokhon duita user communicate korbe, tokhon gateway ke bujhte hobe kon connection diye kon user ke message pathabe. Ejonno ekta Session Service lagbe, jeta ekta connectionID ar ekta userID store kore, ja diye bujhay kon user kon connection use kortese.

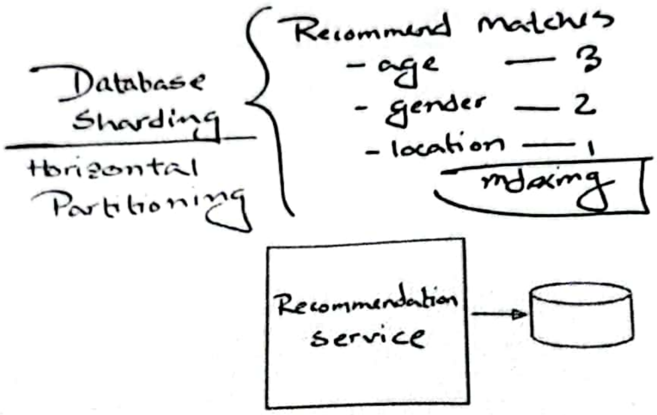
User1 ekta message pathailo, then gateway session service er kache dekhbe ei user1 je connectionID diye message pathaise, shei connectionID ta ar kon user use kortese. Emne user2 er sathe ekta connection establish korbe for the first time. Erpor theke ar connection establish kora lagbe na, oi connection diyei message pass korbe always.



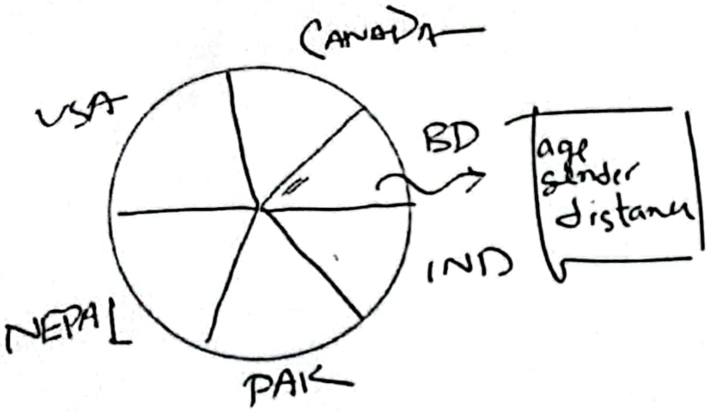
Ei Session Service e message pathanor age Matcher Service use kortese. Match na korle message pathaite dibe na.



Now it's time for Match Recommendation uWu -

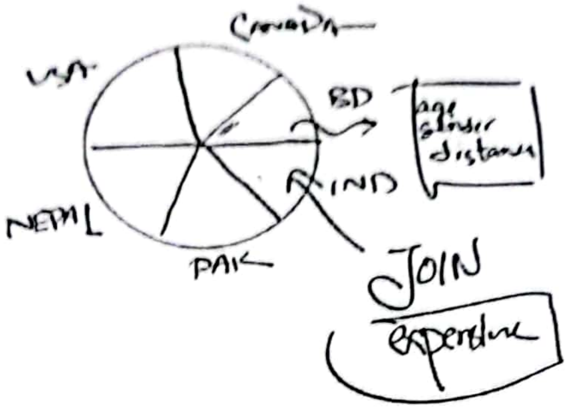


Bangladesh theke pura world er billions of users theke search korte to jaan bair hoya jabe.



Here is a concept, amra BD er jonno only BD er database theke search korbo.

But what if ami border area te thaki?



Tokhon dui region er database er moddhe JOIN request pathaite hobe which is very expensive. But kisu korar nai, it is what it is!

What if ekta user Bangladesh theke India chole jay?

Ei khetre sathe sathe database update korar luxury amader nai. So ekta fixed interval porpor region gular Recommendation Service er database update korte hobe, kono new user ashse kina. This method is called **Database Sharding.**