Push Notifications

- ➔ App ta device e running nai, but still notification ashtese. (Like Bkash)
- ➔ Triggered in 2 ways -
  - ◆ Can be triggered locally by an open application
  - ◆ Can be pushed from server to user when app is not running
- ➔ Push Notifications depend on 2 APIs -
  - ◆ **Notification API**
    - App open thak ba close thak, jekono notification display te show koray.
      (Will discuss more further)
  - ◆ **Push API**
    - Ekta service worker thake. Service Worker hocche ekta script jeta shob shomoy browser e run korte thake, eta khubi low amount of resources use kore, and shob shomoy listen korte thake kono push message/notification ashse kina. Push message ashle sheta Notification API er maddhome user ke show kore.

Important Terms

- **Notification**
  - A message displayed as pop up to user.
- **Push Message**
  - A message sent from server to client.
  - Shob Push Message kintu Push Notification na.
    - Push Message ta ke Notification API er help niye user ke display korle tokhon shetake Push Notification bolbo.
- **Push Notification**
  - A notification created in response to a push message.
- **Notification API**
  - ***Covered above***
- **Push API**
  - ***Covered above***
- **Web Push**
  - Ei pura process e server theke client e message pass korbo. Ei pura process tay je je component involved thake, ei pura jinish ta ke boltesi web push. Ei pura...arre beda dhet.
- **Push Service**
  - Ei service ta; the service/system for routing push messages from server to client.
  - Prottek browser er push service alada, and ektai. It's like a server!
    - For example, amar kache onekgula browser ase. Ekhon ami chrome theke facebook notification

on kore disi. Ami shudhu Chrome thekei tokhon facebook er push notification gula pai. Kokhono emon hoy na je onno browser thekeo same notification gula ashtese.

- **Web Push Protocol**
  - Describes how an application or server interacts with the push service.

➔ ***What can we do with a push notification?***

- Alert user to an important event.
- Display an icon and a small piece of text that the user can click to open up your site.
- Integrate buttons, sounds, vibrations etc.

➔ **Restrictions of Push Notifications on Web**

- Browser e push notification gula new, not advanced compared to Android, iOS.
- Ekek browser ekekvabe push service implement korse. Shob browser shob push functionality support kore na, like sound, vibration nai. Web er khetre ami jani na amar application ta kon browser e use hobe, but android/iOS er khetre sheta jani.

How A Push Notification Works

Here, Server -> Amar app er server

Client -> User er Browser

Notification API

- ➔ Displays notification to the user
- ➔ Can be divided into -
  - ◆ **Invocation API**
    - Controls how to make your notification appear.
    - Just notification ta show korbe.
  - ◆ **Interaction API**
    - Controls what happens when user interacts with it.
    - Notification tar sathe interact korle (e.g. swipe, click) ki ki hobe, sheta handle korbe.

How to Display Notification

- ➔ Notification display korar jonno server sider er kono kaj nai, puratai device / client side er kaj.
- ➔ 3 steps -
  - ◆ **Request Permission**
    - User er kach theke permission nibe

```
Notification.requestPermission(function(status) {
    console.log('Notification permission status:', status);
});
```

  - ◆ **Display Notification**
    - ***Without Options***
      - User er permission ase kina check korbe. Permission thakle service worker ke diye notification show korabe.

```
function displayNotification() {
  if (Notification.permission == 'granted') {
    navigator.serviceWorker.getRegistration().then(function(reg) {
      reg.showNotification('Hello world!');
    });
  }
}
```

without Options

- ***With Options***

```
function displayNotification() {
  if (Notification.permission == 'granted') {
    navigator.serviceWorker.getRegistration().then(function(reg) {
      var options = {
        body: 'Here is a notification body!',
        icon: 'images/example.png',
        vibrate: [100, 50, 100],
        data: {
          dateOfArrival: Date.now(),
          primaryKey: 1
        }
      };
      reg.showNotification('Hello world!', options);
    });
  }
}
```

time in ms

Hello world
Here is notifi body

even indices – vibrate
odd indices – silent

with options

  - Vibrate (100, 50, 100)
    - Even indices - Vibrate
      Odd indices - Silent
    - Ekhane first 100 ms vibrate korbe, next 50 ms silent thakbe, then next 100 ms abar vibrate korbe.
  - **Primary Key**
    - Notification er sathe user er behaviour lock kore rakhte chaile.
    - Suppose, ekta notification 900 jon ke pathailam, 800 jon close kore dilo, baki 100 jon interact korlo. Prottek notification eri nijer

primary key ase. Sheta diye server e log rakhte pari, in order to research je kon notification kon user ke dekhabo ba dekhabo na. Also custom user er jonno custom notification banaite pari.

- **Listen for Events** <------ Service Worker
  - Service Worker notification er jonno listen korte thake, jokhon notification ashe tokhon sheta display kore dey.
  - Notification display korar por service worker listens for events (notification close, key value notification). Then based on that, service worker oi event ta handle kore.
  - Erpor ghumay jay.

Push API

➔ **How it works**

- Push Service
  - Prottekta browser er nijeder push service ase.
- User grants permission
- Subscribe app to browser's push service
  - Client request accept korle client er push service server ke ekta object pathay. Object e thake -
    - **Endpoint URL**
      - Server theke client er browser e kono notification pathate chaile ei URL e pathate hobe. Ei URL er last e ekta unique identifier thake, jeta diye message ta kono ekta particular user kei pathaite parbo.
    - **Public Key**
      - Server jokhoni client browser er push service e kono notification pathabe, tokhon ei public key diye encrypt kore dibe.


```
{"endpoint": https://fcm.googleapis.com/fcm/send/dpH5lCsTSSM:APA91bHqjZxM0VImWWqDRN7U0a3AycjUf4O-byuxb_wJsk
"keys": {
  "p256dh":"BLQELIDm-6b9Bl07YrEuXJ4BL_YBVQ0dvt9NQGGJxIQidJWHPNa9YrouvcQ9d7_MqzvGS9Alz60SZNCG3qfpk=",
  "auth":"4vQK-SvRAN5eo-8ASlrwA=="
  }
}
```

Let's wrap up the whole fking thing -

<u>On the Client:</u>

- ➔ Subscribe to the push service
- ➔ Send subscription object to the server
  - ◆ Endpoint, public key

<u>On the Server:</u> (Application, not push service)

- ➔ Generate the message that we want to send to user
- ➔ Encrypt the data with public key
- ➔ Send data to the endpoint

After finishing all these, browser e jinish ta jawar por -

- ➔ Wakes up the service worker
  - ◆ Receives the message
  - ◆ Show the notification
  - ◆ Handle interaction
  - ◆ Sleep

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.

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.

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

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.

Store profiles Done!

Let’s note down matches now -

Table e userID A, userID B thakbe. That means A, B er sathe match korse. Why is it necessary? Bcz ekbar jehetu match korse, ekhon ami ar A ke B er kase recommend korbo na.

Note down matches Done!

Now let’s implement direct messaging -

Age amra client-server protocol (e.g. HTTP) use kortam. User server e request pathaito, then server response korto, server nije theke user ke data pathaito na. But in this

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**.

Designing Instagram

Features

- ➔ Store/Get Images (File)
- ➔ Like/Comment on Images (Post)

Does it support recursive comment? (1 Layer)

Like on comments?

- ➔ Follow someone
- ➔ Publish a News Feed

CDN (Content Delivery Network)

In this case, Round Trip Time onek beshi lagbe. Cz every static content (e.g. image) load howar jonno main server USA theke data load kora lagbe.

In case of CDN,

Onekgula region e vaag kore nibo. Eder sathe proxy server thake. Jokhon kono static content load kora dorkar hoy, tokhon ei proxy server theke load hoy.

Advantages of CDN

- Performance
  - Round trip time kom lagbe.
- Less load
  - Age main server e millions of request ashto. Kintu ekhon jotogula proxy server, totogula request ashe.
- Security
  - CDN is provided by Amazon, Cloudflare etc. Oder server gular alada firewall ase.

Static Files

- ➔ Image, Video, Pdf
- ➔ TTL (Time To Live)
  - ◆ Suppose, TTL 7 days hoile 7 days por proxy server main server ke request pathaye bolbe, "accha amar kache to ekta image chilo jeta ami 7 din age nisilam, oita ki ekhono same ase naki change hoise? Change hoile notun version ta amake pathay dao."

Two types of CDN -

Push CDN

- ➔ Main server Proxy server ke notun content pathabe jokhoni kono ekta content change hoy. Mane user kono kichu change korlei Proxy server ke Main server update kore dibe, "tomar under e jei user gula ase, tar content e ei ei change ashse."
- ➔ **Advantages**
  - ◆ Notun content gula shathe shathei receive korte partesi.
- ➔ Suitable for -
  - ◆ Sites with small traffic. Server e load kom thakle time to time proxy server ke update korte pare.
  - ◆ Static files frequently update

Pull CDN

- ➔ Fetch new content from server when first user requests it.
- ➔ TTL determines for how long content is cached.

- **Advantages**
  - Main server er load kome jacche. Age ekta content change hoilei main server ke ekta response pathano lagtesilo proxy server ke.
- Suitable for -
  - Sites with high traffic.
- **Disadvantages**
  - Suppose ekta image er TTL 7 days. But user oita 5 days por delete kore dilo. Still proxy server e image ta aro 2 days theke jabe, and oi region er onno user ra shei expired content dekhbe.

Disadvantages of CDN

- Cost
  - Age data gula ekta server e rakhtam. Ekhon alada alada region e server e rakhte hocche. Increases cost.
- Require Changing URLs
  - Common user er khetre Image gula Regional Proxy Server e cached kore rakhtesi. But celebrity er khetre content gula multiple proxy server e store korte hocche; waste of space. Also ekta image er jonno different URL generate kora lagtese.
- Expired Content
  - Suppose ekta image er TTL 7 days. But user oita 5 days por delete kore dilo. Still proxy server e image ta aro 2 days theke jabe, and oi region er onno user ra shei expired content dekhbe.

Let’s get back to designing Instagram.

Store/Get Images is done by creating CDN.

Now let’s design a database for Instagram -

Amar eshob like comment ke koyta korse eshob bishoye kheyal rakhte hocche. Ekhon suppose, keu ekjon post dilo -

Ei post e jokhon keu like dibe -

Keu jodi like remove kore dey, tokhon active field ta false kore dibo. ParentID diye define korbo eta post er like naki comment er like.

Erokom khetre like ta post er naki comment er sheta janar jonno whole id er upor ekta string operation korte hobe, which is very expensive. As a solution, we can add an extra field -

Ekhon jodi jante chai, ekta post e kotogula like ashse -

~~*Select count(*) from Likes where parentID=" "*~~

But in case of Instagram, this is not feasible. Cz proti second e millions of users news feed refresh kore. Shobar jonno database e emne request pathaite gele database dead.

Rather we can maintain a separate table for counting likes and comments for a particular post.

Ekta like korle Likes +1 kore dibo, abar ekta comment korle Comments +1 kore dibo.

Likes and Comments table same.

For followers -

Now let's move on to core Instagram design -

User Gateway server er kache username, password, token etc pathaye authenticated hobe, othoba user1 user2 just message passign korte pare. Gateway server er kaj hoilo user ke authenticate kore deya, ba amar request ta kon service er kache jabe shekhane route kore deya.

Jehetu Gateway server internal server gular sathe communicate kore, tai ekta internal protocol use korte hobe, ja outside er kono server bujhbe na.

Load balancer decide korbe kon user kon News Feed service er kache jabe.

Ekhon dhorlam, user ke news feed e 20 ta kore posts dekhabo. News Feed service, Follow service ke userID pathabe. Oi user er follower der ID Follow service News Feed service ke diye dibe. Erpor Post service ke ei set of userID gula pathale Posts service oi user gular posts News Feed service ke return korbe, and then news feed e oder post gula show korbe.

There can be more services like these -

Optimization

- Limit kore dibo je, jokhon news feed refresh er request korbo, tokhon maximum 20 ta post nibo.
- News feed e jei 20 ta post dekhabo, shegula precompute kore rakhte pari.

For example, 4 ta user er jonno ekta queue maintain korte pari. (Ei user gula kinda ekjon arekjon ke follow kore type kisu ekta!) User 2 jokhon news feed refresh korbe, tokhon baki dero queue update hoye jabe. Next time, jokhon oi user gula news feed refresh korbe, tokhon queue er posts show kore dibo. Etai hocche News Feed Precompute kore rakha. As a result, server er upor load onek kome jacche.

Ei precompute dui jayga te kore rakhte pari - Cache & Database. Which is better? Jei user gula frequently news feed refresh kortese, tader ta cache te rakhbo. Frequent user na hoile Database e rakhbo.

➔ **Notification**

Suppose, user kono post dile tar follower der kache push notification pathabo. But what if it’s Kylie Jenner? Celebrity der million of followers ke eksathe pathaite parbo na. Solution?

- **Batch Notification**
  - Batch batch kore pathaite pari, e.g. every 10 second 10k notification pathabo, next 10 sec abar 10k notification pathabo.
- **Polling**
  - Amar device ta proti 1 mint por por check/poll kore kono notification ase kina. Ami only tokhon notification pathabo jokhon user ta poll korbe. Offline thakle chinta kora lagtese na, online ashle pathabo.

Deployment

1. Push changes or updates from one environment to another.
2. **Environment**
   - For example, amra jekhane code kori sheta Development Environment. Abar je environment theke client amar software/website ta access kortese, sheta Production Environment / Live Environment.

Left to Right Deployment Model

➔ Coding theke shuru kore Client er kache jei process er moddhe diye jabe.

Local → Development → Staging → Live

left to right deployment model

➔ **Local Environment**

- Emon ekta environment setup korsi je amar PC theke message pathai, sheta amar phone e show kore. Ei whole thing tai local environment.

➔ **Development Environment**

- Amar jei messaging service ta (not whole app) local e thik moto run korlo, etake ekta development server e upload korlam.
- In most of the cases, Local and Development environment are the same.

- Might not contain all the microservices.

➔ **Staging Environment**

- Age jeta upload dilam, oita whole app er ekta part. Erokom onek gula service thakbe. Shob gula eksathe Staging Environment e upload dibo.
- It is actually the exact replica of the Production Environment.
- Contains all the microservices + newly added services.
- Here, QA people will do the testing.

➔ **Live Environment / Production Environment**

- Staging e jodi thikthak moto kaj kore, taile app ta Live / Production environment e deploy korbo.

Ei local theke live prottek environment ei jokhon ekta theke arektay changes niye jabo, etai deployment.

Deployment Process Flow

1. Software deployment plan
2. The actual development
3. Testing your changes
4. Deploying changes to the live environment
5. Monitor your changes

Different Types of Deployment

➔ **Deployment of meta-data**

- Codes, css etc
- Always controlled by developers.

➔ **Deployment of content**

- Photos, videos etc

- Content creators can upload.

Deployment Best Practices

- Use Git
- Work in branches
- Review the differences before going live
- Deployment schedule
  - Less number of active users, enough developers - eishober upor base kore ekta deployment schedule choose korbo.
- User groups with different permissions
  - Shob developer ke deployment er permission deyar dorkar nai.
- Stay calm, even if something breaks

Software Deployment Methodology

Devops

- Methodology, set of best practices
- Primary Goals -
  - Shorten delivery times
  - Faster development
- 7 steps -
  - Coding
  - Building
  - Testing
  - Packaging
  - Releasing
  - Configuring
  - Monitoring

- Depends on two frameworks -
  - **Continuous Integration (CI)**
    - New code is integrated in a shared repository on a regular basis. Sometimes it can be several times per day.
    - Proti week / proti din new code repository te push kori.
    - After the integration, the new code should be tested through an automated build process.
  - **Continuous Deployment (CD)**
    - Software Release Strategy ta ekta automated test er bhitor diye pass korbe before releasing into every production environment.
    - Jokhoni kono change ashbe, ei whole change ta ekta whole setup environment er moddhe diye jabe with multiple sets of tests. Ei test er pore sheta production environment e push korbo.

Release vs Deployment

- Ekta exisitng code ase with version number 4.0.1. New feature implement kore ekta version number dilam 4.0.2. This is called Software Release.
- Ei 4.0.2 version ta jokhon live ba kono ekta environment e push korbo, tokhon etake bolbo Software Deployment.

System Design - Netflix

1. **Open Connect** (CDN) for static content
2. **Backend** (AWS Cloud)
   - Microservice

3. **Client Side** (Desktop, Mobile, Gaming Consoles)
   - Build with ReactJS

     Because -
     - Start up time
     - Run time performance
     - Modularity

Amazon ELB (Elastic Load Balancing Service)

- Two tier balancing schema
  - Across zones
  - Across instances
- Zone wise
  - DNS based
  - For example, BD-3 server e zone 0,1,2 thakbe. Each zone e abar multiple instance of EC2 thakbe. Load balancing er khetre first kaj hocche - kon zone e jabe, then oi zone er kon instance e jabe. (Twisted Load Balancing)
- Amazon ELB server gula ke traffic basis e scale up / scale down korte parbo (automatically).

Transcoding

- Optimizing video for different clients
- Ekta movie file er approx. 1200 ta combination hoye database e store thake, eder protteker codec and resolution er combination different.

User kon device use kortese ba user er net connection kemon tar upor base kore user ke movie send kori.

➔ Ekta worker er pokkhe 50 gb movie convert kora huge task. Tai movie gula chunk chunk kore database e store kora thake. User ke pathanor shomoy kono vabe chunk gula merge kore pathano hoy.

Same movie er 1200 ta instace Open Connect server e pathay dibe. Ei Open Connect server gula end user/client ke provide korbe.

Netflix Zuul

Netflix chunk division based on time kore na. Bcz kono ekta important scene e hothat buffer kora shuru korle eta bad user experience. Rather Netflix chunk divide kore Scene Recognition er upor. So that kono important scene cholakalin user buffering face korbe na.
Netflix user ke kokhono 1 ta chunk pathay, kokhono multiple chunks pathay. Netflix e je movie/series/episode/scene ta dekhtesi shetake Netflix 2 ta category te vaag kore -

- **Sparse**
    - Ei type er movie gula te continuous dekhar agroho user er thake na.
    - Ei type movie te user jei chunk er jonno request pathabe, just shei chunk tai dibe.
- **Dense**
    - Ei type movie te user ra jei chunk er jonno request pathay, tar porer besh koyekta chunk dekhe. (Multiple chunks)

Netflix Zuul

➔ Frontdoor/gateway service for all requests from devices to the backend
    ◆ Gateway -> Je request gula ashe, shegula multiple service er kase route kore

➔ Services provided by Zuul -
    ◆ **Authentication and Security**
        - Jokhoni kono request ashbe, sheta authenticated user er kach theke ashtese kina

- **Insights and Monitoring**
  - Per second kotogula request ashtese, ekhon je EC2 instance gula ase shegular health kirokom (health define kora hoy ekta EC2 instance er kase jotogula request jacche, and jotogula successful response ashtese tar ratio theke)
- **Dynamic Routing**
  - Kon microservice system er kase jabe?
- **Stress Testing**
  - Zuul EC2 gulay dummy request pathaite thakbe and health check korbe, kotokkhon successful response back korte pare!

- **Load Shedding**
  - Allocating capacity for each type of requests
  - Dropping requests that go over the limit
- **Static Response Handling**
  - Kichu service jetar jonno kono microservice er help neyar dorkar nai, ei information gula amiee diye dite pari. Egular jonno request gula kono internal cluster e route na kore nije thekei response ta pathay dei.
- **Multi-region Resiliency**
  - Jokhon kono request ashbe, shei request take kon AWS region (bangladesh, india, pakistan

etc) e pathabo, shei region e request ta route kore dey.

Netflix Hystrix

➔ Another algorithm developed by Netflix

Suppose X ekta microservice, jeta abar X1, X2, X3 er upor dependent. Ekhon X3 theke response na ashle X2 o response pabe na, X2 na paile X1 o pabe na, emne last e user o response pabe na. Ei total failure ta ke bola hoy Cascading Failure. Eita detect korte onekgula stage paar howa lagtese; waste of time.

To remove Cascading Failure -

Zuul theke Recommendation Service e je function call ta ase, sheta ekta Circuit Breaker e wrap kore dey.

3 states -

➔ **Closed**

- Connection ase

➔ **Open**

- Connection nai

- **Half Open**
    - Open state er ekta timeout thake. Suppose, timeout 5 min. Ei 5 min por Open theke Half Open state e chole jabe. Ei state e Recommendation Service e ekta request pathay, oita jodi fail kore taile abar Open state e chole jabe and abar 5 min thakbe. Ar jodi fail na kore taile Closed state e back jabe.

Suppose, 10000 requests er moddhe 1 ta request fail korlo, that’s okay. But jokhon 100 ta fail kore ba number of failed requests threshold amount cross kore, tokhon circuit breaker ta closed theke open state e chole jabe. As a result, Recommendation service related ar kono request ashlei Zuul direct fail alert diye dibe, oita X e ar pathabei na, kono kisu check o korbe na.

Consistent Hashing

Hashing

ReqID ta hash kore ekta number pabo, oitake jekono ekta number M diye mod korbo, mod korar por jei remainder ta thakbe shetar upor base kore different server e pathabo.

Not necessarily the number of servers is **M**, it may be **N** !

Consistent Hashing

Array ta ring er moto thake.

hash(ReqID) % M

ReqID hash kore ring er moddhe distribute hoye jacche (based on mod).

No. of Servers

N

hash(0) % M

hash(1) % M

2

3

Server number (0,1,2…) thake, ei server number ba server id gulakeo hash korbo.

Ebar egula circle er moddhe map korlam.

Consistent hashing e kon ReqID kon server handle korbe?
Clockwise jabo, ekta ReqID tar porer server ta handle korbe.

Age cache data server 1 e store hoito. But jokhon new server 5 add korlam. Tokhon ager shob (suppose 80%) cache data 5 e store hocche, and erporer baki 20% cache data server 1 e store hocche.

Eije ager 80% je cache data server 1 e chilo, sheta ekhon ar use hocche na (bcz shegula server 5 e niye gesi), so ei 80% cache data waste. Ar overall e 20% cache data waste. Bcz overall dhorte gele server 5 add korar por 20% cache data server 5 e jacche na.

But there is a problem,

Ekhane ekta fail korle request er porer server e jacche. Emne ekta baade shobgulai fail korle shob request oi ekta server ei jacche. So new server add / kono server fail korle load fairly distribute hocche na.
Age 1-hash function use kortam, ekhon K-hash function use korbo.

1st server er instance gula 3 ta hash function diye hash korbo.

Evabe 2nd 3rd server er instance gulakeo 3 ta hash function diye hash korbo.

Ekhane ek region er load gula tar porer server (here, server-1) e jabe.

Age ekta fail korle always tar porer ekta server er uporei load porto. But ekhon ekta fail korle porer blue server ba white server e jacche. So load gula uniformly distribute hocche.
Abar jokhon server add korbo, tokhon just ekta server er ekta instance na, multiple instance add korbo.

Ei multiple instance gula mulitple server theke load shoray nicche, not just ekta server theke load shoray nicche.

For System Design, we have to consider -

- **Features**
    - What features to add
- **APIs**
    - Microservice / Monolith
- **Availability**
    - **CAP Theorem**
        - **C**onsistency
            - User jokhon request kortese, she shobcheye update write ta pabe
        - **A**vailability
            - User jokhoni kono data er jonno request korbe, sheta available thakbe
        - **P**artition Tolerance
            - Kono distributed system er koyekta node er moddhe ekta node jodi connection loss

hoye jay, tobuo jodi oi cluster ta successfully response dite pare, tahole sheta partition tolerant.

- Amar system eksathe consistent, available, partition tolerant - 3 ta eksathe hoite parbe na. Maximum jekono 2 ta hoite parbe. For example, a phone cannot be cheap, fast and good at the same time.

- **Latency / Performance**
  - Etar upor depend korbe CDN use korbo kina, server spec up korbo kina
  - **SPOF**
    - Single Point Of Failure
  -

    Ekhane server, database duitai SPOF

- ***How to remove/avoid SPOF?***
  - **Load Balancing (More Servers)**

  - **Master - Slave**

For example, ekta database e write korte parbo na, shudhu read korte parbo.
Ekhon ei overall system tay SPOF hocche Load Balancer.
Ejonno multiple load balancer use korte pari.
Ekhon whole system tai SPOF.
Ejonno ekhon whole system ta ke multiple region e store kore rakhbo.

- **Multiple Region**

DNS server dibo, ekekta region e request ta route korbe, based on ip address.

- **Scalability**
- **Durability**
- **Class Diagrams**
- **Security & Privacy**
- **Cost Effectiveness**
  - Ami 3rd party storage system use korbo, naki CDN use korbo, naki multiple region e amar content gula rakhlei hocche, naki whole server e rakha effective? Egular basis e amar logic daar korabo.

An example of System Design -

Ekta TikTok design korbo. Ekhane videos gula rakhar jonno Amazon S3 use korbo. User Data (user er followers, cumulative views etc) change howar probability onek kom, so relational / SQL database e store korte pari (ba NoSQL o use korte pari if I have logic). Video metadata change hoy, so etar jonno kono Key Value database (like NoSQL) use korbo.

Back of The Envelope Calculation

- Storage
- Cache
- Processor

Need to memorize -

1 million $= 10^6$

1 B $= 10^9$

1 Trillion $= 10^{12}$

1 KB

1 MB $= 10^3$ KB

1 GB $= 10^3$ MB $= 10^6$ KB

1 TB $= 10^3$ GB $= 10^9$ KB

1 PB $= 10^3$ TB $= 10^{12}$ KB

avg $\boxed{\text{1 million / day} = 12 \text{ req/sec}}$

User 1 million, per day 1 ta kore request dile avg 12 req/sec ashbe. Similarly, 1 million user per day 10 ta kore request dile 120/sec, 100 ta dile 1200/sec

Peak Time

1 hr / 10% = 100K

30 /sec

Peak time e 1 hour e 10% request (100k) ashe means 30/sec. Similarly, peak hour e 30% request ashe means 90/sec

Numbers every CS E... should know

Latency Comparison Numbers (~2012)

L1 cache reference	0.5 ns		
Branch mispredict	5 ns		
L2 cache reference	7 ns		14x L1 cache
Mutex lock/unlock	25 ns		
Main memory reference	100 ns		20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy	3,000 ns	3 us	
Send 1K bytes over 1 Gbps network	10,000 ns	10 us	
Read 4K randomly from SSD*	150,000 ns	150 us	~1GB/sec SSD
Read 1 MB sequentially from memory	250,000 ns	250 us	
Round trip within same datacenter	500,000 ns	500 us	
Read 1 MB sequentially from SSD*	1,000,000 ns	1,000 us	1 ms ~1GB/sec SSD, 4X memory
Disk seek	10,000,000 ns	10,000 us	10 ms 20x datacenter roundtrip
Read 1 MB sequentially from disk	20,000,000 ns	20,000 us	20 ms 80x memory, 20X SSD
Send packet CA->Netherlands->CA	150,000,000 ns	150,000 us	150 ms

Notes

1 ns = 10^-9 seconds

SF → NYC — 40 ms

SF → AUS — 183 ms

YouTube

Storage

DAU → 1B (viewer)

Viewer : Uploader → 1000 : 1

Daily Uploader = $\frac{10^9}{10^3} = 10^6$ (1M)

Avg Upload per day → 10 minutes

Total upload/day = 10^7 minutes ~ $\frac{166{,}000 \text{ hours/day}}{720{,}000 \text{ hours/day}}$

DAU -> Daily Active User

Estimate kore result pailam 166,000 hours/day, kintu actual result 720,000 hours/day. As long as amar result 10 times range er moddhe ase, I am good to go!

2 hrs unprocessed video = 4GB

processed = 400 MB

1 hr = 200MB

1 minute = $\frac{200}{60}$ = 3MB

Per day koto MB store kora lage?

$10^7 \times 3$ MB = 30TB

10^6 MB = 1TB

Raw Footage

Eta hocche ekta copy. But ami at least 3 ta copy store korbo (Asia, Europe, America). So that CDN er moto kaj korbe

(response time komabe), SPOF remove korbe (data jeno loss na hoy).

3 copy x30TB

= 90TB/day

Another problem is,

720p → x MB

↳ 480 — x/2 MB

360 — x/4

240 — x/8

144 — x/16

} x MB

2x MB

So now the calculation should be -

3 copy x30TB

= 90TB/day

= 90*2 TB

= 180TB

= 0.2 PB

2hrs ~ 4GB

Based on 2hrs~4GB calculation, we came to the result that, per day Youtube has to store 0.2 PB

Youtube DAU -> 1B

Facebook DAU -> 1.10B

Instagram DAU -> 200-500M

(Exam e exact dhora lagbe na, motamoti 10 times er moddhe kisu ekta dhorlei hobe!)

What if one Allah'r banda calculates like this -

1 min = 60 s

Avg. video = 24 fps

1MB

60×24×1 MB

1500 MB = 1.5 GB

1 minute of processed video

And get result -

100PB/day

But should Tasnim bro give him marks? Tasnim bro says, "NO! Jar estimation bhalo joto, s/he gets marks toto!"

Cache Requirements

Ki ki data cache rakhbo?

Let's estimate cache for thumbnail -

Thumbnail actually jeta upload kora hoy, tar theke height-width-size komay ana hoy.

Shob video er thumbnail to Youtube cache te rakhbe na. Number of popular videos + evergreen videos cache te rakhbo. But er number koto? Estimate korlam, 90 days e Youtube e jotogula videos upload deya hoy tar shoman.

According to our estimation, per day uploaded video = 10^6

90 days

Total Video

$\cancel{90}^{100} \times 10^6 \times 10$

$10^3 \times 10^6 = 10^9 KB$

$= 1TB$ in RAM

90 days (90 ke 100 dhorte pari) x 10^6 per day uploaded video x 10KB thumbnail per video.

There are 8 gb per stick, 16 gb per stick (has 64 nodes), 32 gb per stick (has 32 nodes). We choose 16 gb per stick.

8, ~~16~~ ~~32~~ ~~64~~ 32 node

→ 64 Node

How did I calculate?

16 GB X 64 = 1024 GB

Ekhon cache te 3 ta continent (asia, europe, america) er jonno 3 ta copy rakhbo, and also because data jeno loss na hoy (so, 64x3). Then, prottek ta RAM e 100% capacity te data store korte chai na, 50% capacity te data store korte chai, ejonno number of nodes bere jabe (so, 64x3x2).

$$16\ GB \times 64 = 1024\ GB$$

$$64 \times 3 \times 2$$

384 Nodes

So, Youtube e total cache store korar jonno 384 nodes lagbe. Shob gula node 50% capacity te. Prottek ta node er size hocche 16 GB. Etai hocche Youtube er Cache Requirements.

No. of Processors

Per day koto minute er video process kora lagbe?

10^7 minutes of video / day

But,

$$\frac{\text{No. of } \underbrace{\text{Processors}}_{MB}}{\underbrace{10^7 \text{ minutes of}}_{\text{Time}} \text{ video / day}}$$

Somehow, ei Time ke MB te convert korte hobe.

Let,

$$1 \text{ hr unprocessed data} = 1 \text{ GB}$$

Now, 10^7 minutes = 167×10^3 hours

$$167 \times 10^3 \text{ hours}$$

$$167 \times 10^3 \text{ GB}$$

Eta ekhon /day te ase, /sec e nibo.

$$167 \times 10^3 \text{ GB/day}$$

$$= \frac{60 \times 3 \times 10^3}{24 \times 3.6 \times 10^3} \text{ GB/sec}$$

1 day = 86,400 sec = 24 x 3600 sec

$$= \frac{60 \times \cancel{3} \times \cancel{10^3}}{24 \times \cancel{3.6} \times \cancel{10^3}} \text{ GB/sec}$$

$$= \frac{60}{24} \text{ GB/sec} \approx \boxed{2 \text{ GB/sec}}$$

process

Approximately 2GB/sec amount of video process korte hobe.

Ekhon ei processing korte kotogula computer, node, processor lagbe, sheta calculate korbo -

Ekta video process korte 3 ta step follow korte hobe -

Read + Processing + Write

Let's now estimate,

Read + Processing + Write

10ms/MB + 20ms/MB + 20ms/MB

$\boxed{50\ ms/MB}$

Ekhon 2GB process korte,

2000 MB × 50 ms

$\frac{2\times10^{3}}{2GB} \times \frac{50\times10^{-3}}{50ms}$ sec

= 100 sec

So,

1s ⟹ 100s

1 sec e je video generate hoy, sheta process korte lage 100 sec

Now another problem is,

2GB/sec process

multiple resolution
multiple format
10 times
20GB

Multiple resolution, multiple format er video process korte gele overall calculation ta ke 10 times baray dibe.

So,

$$\frac{2\times10^{3}}{2GB}\times\frac{50}{50ms}\times10^{-3}\times10\,sec$$

$$=100\times10\,sec=1000\,sec$$

$$1s \Rightarrow 1000\,sec$$

1000 sec dhore jodi 1 sec er video process kori, tahole porer second er video ta kobe process korbo? So I need 1000 processors.

Processors = 1000

Let's now see an example of ***Instagram***,

Instagram

200 million DAU × 30 photos = 6×10^3 M = 6 B Read Requests

External

200 million users protteke per day 30 photos dekhe (read).

200 M × 0.1 photos upload = 20M write Requests

We know, 1 million req/day = 12 req/sec

So, 20M req/day = 240 req/sec

240 /sec — write

Again, 1M req/day = 12 req/sec

1B req/day = 12 x 10^3 req/sec

So, 6B req/day = 6 x 12 x 10^3 req/sec

$6 \times 12 \times 10^3 = 72{,}000$ Read

So, Read : Write,

$$\boxed{72 \times 10^3 : 240}$$

So, Instagram is Read heavy.

Now let’s calculate bandwidth,

For per day Read,

Bandwidth

$$6B \times 1\cancel{5MB} = 6\times10^{9} \times 1$$

$$= \frac{6\,TB}{86{,}400} \text{ per day}$$

$$= \frac{6\times10^{9}}{10^{5}} = 6\times10^{4}\ mB/sec$$

Similarly, per day Write er jonno 20M x 1MB

Database Sharding/Partitioning

- Ekta system er performance scale up korte chaile last resource howa uchit Database Sharding. Er age ki ki korte pari?
  - Read request e beshi time lagle Query improve korte pari, Join gula remove korte pari
  - Database De-normalization
  - Duplicate Database

  Eto kisur poreo performance improve korte na parle then last way hishebe Database Sharding e jabo.

Vertical Partitioning

- Column wise partition

Horizontal Partitioning

- ➔ Sharding

- ➔ Row 1-100 ek jaygay thakbe, 101-200 arekta jaygay thakbe.

Horizontal/Vertical jekono partition ee 2 type er hoite pare-

- **Physical**
  - Duita part / duita shard different machine e thakbe
- **Logical**
  - Ekta table ke multiple table e convert korbo.
  - Same machine e thakbe.
  - ***Benefit***
    - Faster Query

Jei data ta chacchi, sheta kon shard e ase ta jante hobe. Etar upor base kore 2 vaage vaag kora hoy -

- **Algorithmic**
  - Application e jodi kono function thake jeta use kore shard number ta ber kore felte parbe.

  - ***Advantage***
    - Arekta module er upor depend korte hocche na
  - ***Disadvantage***
    - Sharding mechanism change hoile application abar change kora lagbe
- **Dynamic**
  - Ei information janar jonno application ke jodi onno kono module er upor depend korte hoy

- ***Disadvantage***
  - Arekta module er upor depend korte hocche
- ***Advantage***
  - Application er jana lagbe na ami kon sharding mechanism use kortesi

Partition Criteria

- **Key**
  - Table er jekono kichu ke key fix korbo. Not necessary sheta primary key hote hobe, but unique howa lagbe. Jekono duita column er composite o hote pare.
    Erpor key ta ke hash function e diye number of shards diye mod kore je result pabo, shei DB te data ta store/read korbo.

  - ***Advantage***
    - Balanced Distribution
  - ***Disadvantage***
    - New shard add korle jhamela hobe
      - **Solution:** Consistent Hashing

- **Range**
  - Range example -
    - Date, Price, Genre, Alphabetical Order

      $1 — $100

      $101 — $500

  - ***Advantage***
    - New shard add kora easier. Consistent hashing use na korleo key based sharding er moto problem nai.
  - ***Disadvantage***
    - **Hotspot**
      - Suppose, Birth Month er basis e sharding kortesi. July/Aug range e beshi manush hoye gese. Onno month e kom hoye gese.

Ekhane July/Aug range ta hocche Hotspot.

- **List**
  - Similar to Range based sharing. But set of values thake.

- **Round-Robin**
  - Kono ekta field nei, oitake number of shards diye mod kore dei.

  - ***Disadvantage***
    - New shard add korte jhamela hobe. Ekhane consistent hashing amake [bachate parbe nako]() :(
  - ***Advantage***
    - Hashing is expensive
    - Shob shard e shoman poriman data nishchit hoy (in Lovenone's voice)
- **Location**
  - Tinder poranor shomoy poraisilo :3
- **Composite**
  - Egula theke multiple kisu diye composite criteria banaitei pari

Common Problems of Database Sharding

→ **Joins**

- DB sharding er shomoy at any cost Join Reqeust avoid korar try korbo. Ar Join Reqeust avoid korar ekta way hocche - DB denormalization
- **DB Normalization**
  - Ekta DB te Course table ase, Teacher table ase. Course table ke ekta machine e rakhbo, Teacher table ke arekta machine e rakhbo.

- **DB Denormalization**

        - Course Table theke data dekhanor shomoy shob shomoy Teacher's Name ekta additional information dekhano lagtese. 90% khetre join request diye Teacher table theke Teacher Name neya lagtese. Ekhon ei join requests avoid korar jonno Teacher table theke Teacher Name column ta Course table e ene rekhe dibo. Jodio duplicate column thaktese, but join requests avoid hocche.

- ➔ **Integrity**
    - ◆ Data gula different machine e ase. Tai integrity thik rakha difficult.
- ➔ **Balanced Distribution**
    - ◆ Key based sharding chara kono tatei data balanced distributed hocche na. Kono shard e kom hocche, kono shard e beshi hocche.
- ➔ Database Sharding is difficult. Ekbar shard korar por database ke ager obosthay firay neyao tough. New shard add korao bhejal uff.

Asho ebar ektu onno kichu pori :)

NoSQL database can be of multiple types -

- **Key Value**
    - Ekta key er under e jekono value set korte pari. Value jekono type hote pare.
    - Example :-
        - Amazon S3, Redis

- **Document Based**

  - Key er under e value ta shob shomoy document howa lagbe. Key-value database e kono schema chilo na. But ekhane Key er under e Document tar ekta schema thaka lagbe, like Document ta JSON howa lagbe, emon kisu rule impose korte parbo.
  - Obosshoi key-document gula ekta collection er under e thaka lagbe.
  - Example :-
    - MongoDB

- **Wide-Column Based / Column Based / Columnar**
  - Data gula column wise rakha hoy.

For example, ekhane 1st entry te first column e user1 er ekta data rakhbo, second column e user1

eri arekta data rakhbo, third column eo user1 eri arekta data rakhbo. Abar 2nd entry te user2 er.

- Example :-
  - Cassandra

- **Graph Based**
  - Ekjon post dilo, oi post e jara like dise tader mutual kara - eshob jinisher khetre use kora hoy.

    Normal relational database e ei query gula run korte chaile onek expensive hoye jabe.
  - Data gula rakha hoy 3 ta basis e -
    - **Node**
      - Post hoite pare, User hoite pare
    - **Edge**
      - User and Post er moddhe relation
    - **Property**
      - Post node er property gula different, abar User node er property gula different
  - Example :-
    - Neo4j

Transactions

- DB software may fail at any time.
- The application may crash at any time.
- Interruptions in the network can cut off the application from DB.
- Concurrent Writes, Reads -> **Race Condition**
  - User1 & user2 same time e database er same row te write kortese. Othoba user1 write kortese, user2 read kortese.
- Partially updated data
  - User1 kono data ordhek write kore rekhe dise. Ei data user2 er kase kono make sense korbe na.

Solution: **<u>Transaction</u>**

- Multiple read and write operation ke ekta operation hishebe bind kore fele.

- Ekta transaction ke successful bolbo jodi shobgula read & write operation successful hoy. Ekta transaction successful howa ke boli -> **Commit**

- ➔ Ei read & write operation er moddhe kono subset o jodi unsuccessful hoy tahole amra ki korbo? -> **Rollback / Abort**
- ➔

Advantage of Rollback

- Retry korte pari

Advantages of Transaction

- ➔ Make sure je database e kono partially updated data nai. That means shobshomoy database ta ekta consistent state e thaktese.

Transaction Safety Guarantees

ACID

Apnara jara ACID ke chinen na

Atomicity

- ➔ Suppose user1 database write kortese, and user2 read kortese at the same time. But user1 er write full complete na hoile sheta user2 dekhbe na, that means user2 kono partially updated data dekhbe na.
- ➔ Abar kono fault (e.g. crash, error) occur korar por user read request dile she emon ekta version dekhbe jeta previously committed, emon kono version dekhbe na jeta partially updated.

Consistency

- There are some statements about the data that must always be true. For example, ekta accounting system e always debit ar credit equal howa lagbe.
- A,I,D database vendor dite pare, but C database vendor dite pare na, eta application er upor dependent.

Isolation

Kono database Strong Isolation provide kore, kono database Weak Isolation provide kore. Shob database strong isolation provide korte pare na, bcz isolation er sathe performance cost related thake.

Benefit of Isolation

- Race condition provide kore.
- **Serializable Isolation**
  - Suppose, user1 jokhon ekta data update korar jonno read korlo, tokhon she sheta lock kore dibe. Jotokkhon user1 data ta update kore commit na korbe, totokkhon data ta lock thakbe, and ei locked obosthay data ta onno keu update/read korte parbe na. User1 er update kora shesh hoye gele commit kore dibe, erpor user2 update korte parbe.
  - Shob transaction parallelly handle kore bole database er performance kome jay (performance cost), that’s why shob database vendors eta implement kore na. But still this is the strongest isolation.

Additional performance cost gula jeno na lage, shejonno database vendor ra weaker isolation implement kore.

Durability

- ➔ User er store kora data jeno na haray. For example, single node database e na rekhe multiple node e replicate kore rakhlam.
- ➔ But no system is completely durable.

Single Object Writes

Ekta 20KB JSON file e aro 20KB update korte chai. 10KB update korar por kono problem er karone bakituku update hoy nai. *Now we have some questions need to be answered -*

- Ei 10KB data ki store kora uchit? Naki ager data tai rekhe deya uchit?
- Onno user jokhon ei data ta read korbe, tokhon kon value ta dekhbe? 30KB or 20KB?

We can handle these by -

- **Atomicity**
  - Ekta cache recovery server rakhte pari. Data update start korar age er previous value ta kono ekta recovery server e store kore rakhbo. Jotokkhon update cholbe, totokkhon oi recovery server theke user ke value provide korbo.
  - Ar multiple user same time e jeno write korte na pare, shejonno lock mechanism implement korte hobe.
- **Isolation**

Egulake bole **Light Weight Transaction**. Actual transaction e multiple read & write ke ekta operation hishebe dekhto.

Multi Object Transaction

Why do we need it?

- Suppose ekta relational database e ekta table theke arekta table foreign key reference thakte pare. At the same time multiple table e lock korte hobe, multiple table er user data cache server e niye jete hobe. Single er cheye onek expensive. Egula handle korte multi object transaction dorkar hoy.
- Suppose Relation DB use na kore Document data model / NoSQL use korlam. Document model eo database denormalization er karone emon hoite pare je multiple document e update kora lagtese. As a result, database inconsistent state e pore jete pare. Ejonno document data model eo multi object transaction lagte pare.

Retry mechanism is not always perfect. For example, memory overload er karone kono transaction error khaitesilo, oita barbar retry koreo kono labh nai. For solution, number of retry er ekta upper limit set kore dite pari.

Error can be of 2 types -

Transient Error

- ➔ Deadlock
  - ◆ Multiple write at same time, isolation violation, temporary network interruption etc
- ➔ Ei type er error e barbar Retry korte parbo

Permanent Error

- ➔ Constraint Violation
- ➔ Ei type er error e barbar retry koreo kono labh hobe na. Ekhetre application level e ekta limit set kore dite hobe.

Strong isolation gula always provide kora possible hoy na, bcz of performance drawbacks. So we have,

Weak Isolation Levels

Read Committed

- ➔ A weak isolation method jeta dirty read, dirty write duitai prevent kore
- ➔ **Dirty Read**
  - ◆ Jokhon database theke read korbo, tokhon make sure korte hobe ami only shei data gula read korte

parbo which has already been committed. That means no dirty reads.

- No uncommitted data.
- Example -

→ Dirty Write

- Jokhon database e kono data overwrite korbo, tokhon make sure korte hobe ami only shei data gula overwrite kortesi which has already been committed. That means no dirty writes.
- Partially updated data kokhonoi overwrite korbo na.
- Example -

➔ ***How could we handle this dirty writes?***

- Jokhoni user1 write er jonno access korbe, tokhoni sheta lock kore dibo. Jotokkhon na user1 commit na korbe totokkhon sheta unlock korbo na, so user2 access korte parbe na.

Amra **Weak Isolation** tokhoni bolbo, jokhon task gula parallelly cholbe. Task gula serially cholle boltam **Strong Isolation**.

Buccho Labnan? ;)