The new way to connect

Chat  Moments  Contacts  Search  Pay

1 Billion
monthly active users
WeChat’s Microservice Architecture

- **Service DAG**
  - Vertex: a distinct service; Edge: call path
  - **Basic service**: out-degree = 0
  - **Leap service**: out-degree ≠ 0
    - **Entry service**: in-degree = 0

![Service DAG Diagram]

- **Entry Leap Services**
  - Login, Messaging, Moments, Payment

- **Shared Leap Services**
  - Group Chat, Batching, Notification

- **Basic Services**
  - Account, Profile, Contacts, Msg Inbox
Dealing with Overload

• It’s usually hard to estimate the dynamics of workload during the development of microservices.

Subsequent Overload

How about random load shedding?
Relative Statistics of WeChat Service Requests
• **Overload detection**
• **Service admission control**

• **Requirements**
  – **Service agnostic**
    o Benefit the ever evolving microservice system
    o Decouple overload control from the business logic of services
  – **Independent but collaborative**
    o Decentralized overload control
    o Service-oriented collaboration among nodes
  – **Efficient and fair**
    o Sustain best-effort success rate of service when load shedding becomes inevitable
    o Bias-free overload control
Overload Detection

- **Load indicator of a node:** *Queuing time*
  - Rationale: to manage queue length for SLA

- **Why not** *response time*?

- **Why not** *CPU utilization*?
Service Admission Control

Shuffling on an hourly basis

User Priority

High

1 2 3 4 ... 127 128 1 2 3 4 ... 127 128

Business Priority 1

Business Priority 2

Priority decreases

Login

Payment

Send text message

Send image message

Moments post

Send video message

Moments refreshing, Create group chat, ...

Exploit histogram for real-time adjustment
DAGOR Workflow

- Service agnostic
- Independent but collaborative
- Efficient and fair

Collaborative Admission Control

Node 1
- Leap Service $A$
  - Admission Control
  - ✔
  - ✗

Node 2
- Basic Service $M$
  - Admission Control
  - ✔
  - ✗

Threshold Table
- Node 2, Service $M$ → ($B^*$, $U^*$)
- ...
-...

Entry Service
- User ID
- $B$ $U$
- Sub-request

Response
- User ID
- $B^*$ $U^*$
Overload Detection

Queuing Time vs. Response Time

[Graph showing the relationship between success rate and feed rate for different response times.]
Scalability

Overload Control with Increasing Workload ($M^2$)

Overload Control with Different Types of Workload

Optimal Success Rate = $f_{sat}/f$
Takeaways: DAGOR Design Principles

1. Must be decentralized and autonomous in each service/node
   – Essential for the overload control framework to scale with the ever evolving microservice system

2. Employ feedback mechanism for adaptive load shedding
   – Essential for adjusting thresholds automatically

3. Prioritize user experience
Thank You ALL!