

OneStepOffer系统设计

第四讲

聊天系统与访问限制系统

大纲

- Design WhatsApp : 聊天系统
 - Work Solution
 - Real-time Service
 - Online Status: Pull vs Push
- Design Rate Limiter
- Design Datadog : 一种网站的数据统计, 类似于google anaysis

这节课之后您可以学会

- 设计聊天系统的核心: Realtime Service
- Pull 与 Push 的进一步比较分析

相关设计题

- Design Facebook Messenger
- Design WeChat

聊天系统的使用场景

设计功能

- 基本功能
 - 登录注册
 - 通讯录
 - 两个用户互相发消息
 - 群聊
 - 用户在线状态
- 其它功能
 - 历史消息
 - 多机登录 multi devices

聊天系统的使用场景

系统限制

- 活跃

- 1B 月活跃用户
- 75% 日活跃 / 月活跃
- 约750M日活跃用户

为了计算方便起见，我们来设计一个100M日活跃的WhatsApp

- QPS:

- 假设平均一个用户每天发20条信息
- $\text{Average QPS} = 100\text{M} * 20 / 86400 \sim 20\text{k}$
- $\text{Peak QPS} = 20\text{k} * 5 = 100\text{k}$

- 存储:

- 假设平均一个用户每天发10条信息
- 一天需要发 1B，每条记录约30bytes的话，大概需要30G的存储

Service 服务

- Message Service : 负责管理信息
- Real-time Service : 负责实时推送信息给接受者

storage存储方法

1. Message Table

```
SELECT * FROM message_table
```

```
WHERE from_user_id=A and to_user_id=B OR to_user_id=B and from_user_id=A
```

```
ORDER BY created_at DESC;
```

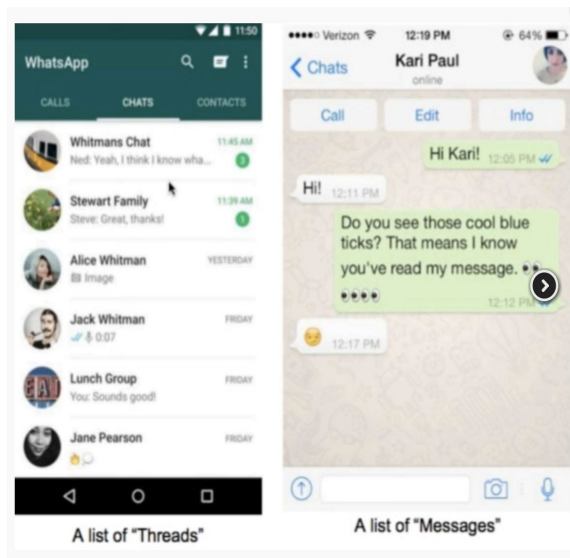
id	int	PK
from_user_id	int	发消息人
to_user_id	int	收消息人
content	text	消息内容
createtime	timestamp	发送/产生时间

只有mesaaage table的缺陷

- 效率太低。
 - a. 如果单纯用sql获取内容，效率太低 -- 可以用kafka 等 message produder 方式解决
- 无法支持群聊功能
 - 群聊需要加入新的表做join

改进 -- 引入thread

hread表示两个的对话如下图所示，左边是thread。右边是message。



Thread Table设计

Message Table		
id	int	
thread_id	int	
user_id	int	谁发的
content	text	发了啥
created_at	timestamp	啥时候发的

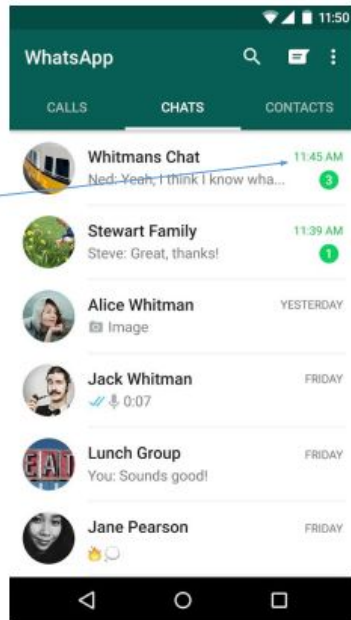
Thread Table		
owner_id	int	谁的Thread
thread_id	int	
participant_ids	text	
is_muted	bool	
nickname	string	
created_at	timestamp	
updated_at	timestamp	index=true

流程梳理

- message table, 存储每个人发的每条消息, 以及对应的thread_id。
即[id, thread_id, user_id, content, created_id]
- thread table, 存储每个thread的信息, 每个人独享一份数据。即
[owner_id, thread_id, participant_ids, is_muted, nickname, created_at, updated_at]
- 查询我的所有thread list——从thread_table里查询给定user_id的所有thread_id即可。因此thread_table用SQL
- 查询我的某个thread的大概信息——从thread_table里查询给定owner_id+thread_id的信息
- 查询我的某个thread的具体消息——从message_table中, 根据thread_id查询所有的消息, 并且返回。因此message_table用NoSQL
- 查询我与另一个用户是否有一个thread?

流程梳理

- Message Table (NoSQL)
 - 数据量很大, 不需要修改, 一条聊天信息就像一条log一样
- Thread Table (SQL) —— 对话表
 - 需要同时 index by
 - Owner ID + Thread ID (primary key)
 - Owner ID + Updated time (按照更新时间倒叙排列)
 - NoSQL 对 secondary index 的支持并不是很好



Word Solution

用户如何发送消息？

- Client 把消息和接受者信息发送给 server
- Server 为每个接受者（包括发送者自己）创建一条 Thread （如果没有的话）
- 创建一条 message (with thread_id)

用户如何接受消息？

- 可以每隔10秒钟问服务器要一下最新的 inbox
- 如果有新消息就提示用户

Scale 扩展

sharding

- Message 是 NoSQL, 自带 Scale 属性
- Thread 按照 user_id 进行 sharding

每隔十秒要inbox?太慢了, 优化——Socket

Socket——TCP长连接

Push Service——提高Socket连接服务, 可以与Client保持TCP长连接

- 当用户打开APP之后, 就连接上Push Service中属于自己的socket
- 有人发消息时, Message Service收到消息, 通过Push Service把消息发出去
- 如果一个用户长期不活跃(比如10分钟), 就可以断开连接, 释放掉网络断开
- 断开连接之后, 如何收到消息?
 - 打开APP时主动pull + android GCM/iOS APNS
- Socket连接与HTTP连接的最主要区别;
 - HTTP连接下, 只能客户端问服务器要数据
 - Socket连接下, 服务器可以主动推送数据给客户端

Scale 扩展流程

- 用户A打开App后, 问 Web Server 要一个 Push Service 的连接地址
- A通过 socket 与push server保持连接
- 用户B发消息给A, 消息先被发送到服务器
- 服务器把消息存储之后, 告诉 Push Server 让他通知 A
- A 收到及时的消息提醒



微信的系统设计

微信的系统设计和whatsapp类似 分为以下几步



图 1 微信消息模型

微信服务的消息队列



图 4 单聊和群聊消息发送过程