Rate Limiter 高频

**Messaging System : Design WhatsApp/Facebook Chat/WeChat**

* Messaging System:
* Scenario:
  + 用户登录注册
  + 通讯录
  + **两个用户互发消息**
  + **群聊**
  + **用户在线状态（在线就撩）**
  + 其他功能
    - 历史消息
    - 多机登录 Multi Devices
  + QPS challenge:
    - 1B 月活跃用户， 75%日活跃用户: 750M 日活跃用户
    - 为了计算方便：我们用100M日活跃用户来计算 （方便起见）
    - QPS：假设平均一个用户每天发20条，那么Average QPS = 100M \* 20 / 86400 = 20K
  + Peak QPS: = 20K \* 5 = 100K --
  + Read heavy 还是 write heavy： 既读又写 （读高写也高）
  + 存储空间：
    - 30G 存储空间。 不重要
* Service:
  + Messaging Service 负责信息管理
  + Real-time Service：负责实时推送给接受者： Push
  + 概念纠正：
    - 手机 <-> 手机 ， 还是 手机 <-> 服务器 <-> 手机:
      * 最简单的 反例：肯定是经过服务器。 因为手机不在线时，依然需要能接受消息（Service）。服务器要存储消息
* Storage：
  + 怎么存储Message：（Message table 存什么）。至于要不要　永久存，还是　存１个月就扔掉，这个是有权衡的，一般会先标记　为　ｔｏ　ｂｅ　ｄｅｌｅｔｅｄ，然后过段时间就删除了。 如果Message table这样存：
    - id, from\_user\_id, to\_user\_id, content，created\_at。 那么看两个人聊天的记录时，需要：SELECT \* FROM message\_table WHERE from\_user\_id=A, and to\_user\_id=B OR from\_user\_id=B and to\_user\_id=A, ORDER by created\_at DESC. Query 特别复杂，效率特别特别低。并且，如果**多人群聊**，那select query就越来越复杂。 怎么办？增加Thread Table “会话\聊天室”的概念
    - Thread 里面 存些什么：
      * 应该是 在在 message 里面 存 其 thread id，因为 一个 thread 对应 多个message。不是一存多 而是 多 存 1而不是 thread里不存很多message id
      * participants，参与者 有哪些，用 serialized JSON 列表
      * created at
      * updated at （用来按照 最后更新的 来进行 倒序排列）
    - Message table里面就要存
      * id
      * thread\_id 把 to\_USER\_ID 变成 thread\_id
      * user\_id 谁发的message
      * content
      * created\_at 啥时候发的
    - 问题：有一些Thread信息是私有的：Thread 名字，静音muted Thread。你把我名字设成A，我可以把你的名字设成B。participantsID 是共享的。每个人可以给别人设置 nickname
      * 所以Thread Table
        + 还要增加：owner id
        + 以及那些私有的property: 群聊名字（is\_muted, Thread）
      * solution: 每个人对 每个 会话 （聊天室）都有一个Thread，共享一个thread\_id
      * 所以对于Thread Table，primary key 是 owner\_id + thread\_id -- 所以此时 primary key 就是
    - 既然每个thread table有重复的信息，要不要单独拉出来一个表：
      * 可以这么做，但是每次用thread的时候都会用到那些私有信息，没有必要。如果login和user info的话，则可以拆开。
      * 这是一个denormalization 的概念，多存一些冗余信息，用来换取更快的访问速度。即便是有重复的数据，只要 能够减少 query 的 效率 就用 denormalization 就没问题了。
    - Message Table读写都比较多：NoSQL -- 可以偷懒， noSQL诞生的原因就是 要记录log，和 message本身 性质很像
      * 数据量很大，不需要修改，一条聊天信息就像一条log一样
      * Thread Id 作为 row key 进行 hash，决定去 哪个 sharding。尽量把同样的Message放在同一台机器（考虑到Scale）。 我们希望 把 同样的 消息 放到同一台机器
      * 写面向对象代码的时候，一个 thread class 包含a list of messages。但是 在 database中存储是反过来的，message中存的 thread id（一对多的时候都是这么玩）
    - Thread Table 需要secondary index，用SQL更好
      * 需要同时 index by
        + Owner Id + Thread id 作为 primary key，找到 所有 owner 当前的 thread
        + Owner ID + Updated Time 按照 时间进行 排序
      * 需要同时按照 owner id + updated time （按照更新时间倒叙排列） 进行index
      * 可能还需要对participantsId进行index，找到之前聊天的记录
      * NoSQL对secondary index支持的不是特别好。
  + Work Solution:
    - 用户如何发送消息：
      * Client 把消息和接收者信息发送给server
      * Server为每个接收者 创建一条Thread (如果没有的话）
      * 创建一条message
    - 用户如何接受：
      * 可以每10s poll一下问服务器要一下最新的inbox
        + 虽然很笨，但是是可行解
  + Scale:
    - Message Table 是No SQL，按照 threadId自动支持sharding
    - Thread Table是SQL，需要用owner\_id 进行sharding。因为每次都去找同一个user。我们希望 界面的 共同属性是什么，都是属于当前 owner，所以用 owner id 进行 sharding
    - How to speed up?
      * 10s 更新信息太慢了
      * solution: 可以用WebSockets。
        + TCP的的**长**连接 方式
        + 和HTTP的区别： client向server发消息，server返回后connection就断了。只能由client向server request。不能由server 推送给 client
        + Socket的话，Server 可以主动给client发消息，因为一直是连着的。服务器可以主动推送数据。如果用户不活跃，可以释放网络端口。手机开机的时候，还要主动poll一下。
        + Socket解决的问题是，服务器可以主动推送给客户端。
      * Push Service提供Socket连接服务，通过Push Service把消息发送出去。如果一个用户长期不活跃，可以断开连接。释放网络端口
      * 断开连接之后，如何收到新消息：
        + 打开app主动pull消息，或者用 Android GCM / IOS APNS Google 或Apple也提供 了类似的Push Notification。这些本质也是 WebSockets，只不过连的是 Google 或者 Apple 的 Service。
        + 如果 是 Web Based，只能重试一下。
      * Flow:
        + 如果B要给A发消息。A 打开 APP，问Web server要一个Push Serer的IP
        + A 通过 socket与push server保持联系
        + 用户B发送消息给A，消息先发送到服务器 （web server）
        + 服务器把消息存储之后，告诉push server通知A
        + A 收到既是的消息提醒 （你有新消息了）： 消息比较短的话，直接push过去，否则如果消息 比较长，可以让A poll一下。
        + Push Service 有好多push server，因为一台机器的 port number 一共只有 65535，需要 用多个 push server 构成push service，sharding by user\_id。让每个 server 的 消息 都对到一个 push service
        + 一边通过自己的service发一条，如果 发不成功，就 用 Android 、 ios 来 push notification。最后实在不行，再让client 去 pull。
        + 如果消息比较多，怎么办？如果 10 年没上QQ了，Pagination／ Infinite Scroll. 只需要 下载 最新的 100条，然后scroll一下，再弄新的。
      * 群聊：如何知道群聊500人大群的人 所有人都能 即时 收到消息
        + 如果一个群有500人，怎么保证 大家都 同时 得到 push server 推送的 新消息。

解决 方案：

Channel Service: 存在于Message Service和Push Service之间。

点对点的聊天，

* + - * + Message service不知道谁在线，谁不在线。收到消息后，发送给 Channel Server。
        + Push server 本身 知道 哪个 connection 还连着。可能通过 socket，也可能通过 apple ios 的 push service
        + Channel Service管理500人在线还是不在线（通过Push Service 有哪些 socket 连接上）。Message Service给Channel Service，所以不需要给所有的都发。这样就不需要push给这么多人，专门用一个service 去做。
        + Channel Service存在内存就好，重要程度不高！
      * 如何check 在线状态：(online status)：
        + 并不是特别精确的查询。
        + 服务器需要知道谁在线，谁不在线（push or pull 是 服务器主动问呢，还是 client 汇报 -- 应该是 汇报，就像 master - client architecture 一样）

push server：服务器主动问 -- No No No

pull： 我主动汇报：（heartbeat的方式）。如果服务器1分钟内没有收到我的heartbeat，就知道我下线了。

* + - * + 用户需要知道我的哪些好友在线

服务其主动发 -- No No No 不需要实时性

pull: 我主动问 -- Yes

* + - * + 实时性特别高的，需要用push，不特别高的，用pull
        + 可以打开Facebook Messenger ，Chrome Network Traffic验证一下，每隔3-5秒 pull 一次服务器。可以告知服务器我在线，同时得知好友谁在线/谁不在线。 既发送了 heartbeat，又收到了在线状态
        + Push虽然快，但是不稳定，客户端网络状况特别不稳定。可能断开。
        + Pull慢，但是稳定。因为只要client在线，server基本一定在线。
      * Push的目的是告诉客户端 有新消息，可以辅助 传回一些数据，真实的数据，还是要通过 用户打开微信，然后重新pull一下。push只是用来优化。poll是比较稳定的。push server 仅仅是 dummy 从 server 发送 消息，没有具体 user 信息。push的时候如果 消息量 比较大，可以提醒 client，you have new messages
      * 点击发送 按键，就是发消息
* Rate Limiter统计系统（用来做限制，login 的时候，会说 请 隔5分钟再来，限制一些行为的次数）: How to limit requests? 如何限制访问次数：一定要问清面试官是考Data Structure 还是 System Design
  + 比如1小时内不能重置 > 5次的密码
  + 如果面试官担心你没有很多system design经验，又想考考你。这其实是一道算法题 + System Design
  + 如果按算法来做的话：
    - A: login time = 0000
    - A login time = 0001
    - 设计一个数据结构 查询最近5分钟 什么东西 发生了多少次。如果把用户的行为存这样一条记录。但是这样就走歪了。其实rate limiter 是 为了限制数据库的 写操作。
  + Rate Limiter 的目的是限制 对 数据库的 读操作。如果按照上面的做法，就把数据库的读操作，变成了数据库的写操作。 很奇怪！！！！！！好像违背了初衷
  + 可以看一些 开源的Rate Limiter 网站实现
  + 谁，在什么时候，做了什么事情。
  + Scenario：（某个 feature，在某个 time，做了某个 event）
    - 无需做到 最近30秒，最近21分钟这样的限制，粒度不需要特别细
    - 只需要限制 比如 2/s, 10/min, 1day，1mon这些general的概念
    - 和面试官聊天， 如果从算法的角度，支持精确的时间范文，从数据结构角度，就需要用segment tree 或者 AVL TreeMap来解这件事情，来求绝对精确的粒度。如果在数据库中，进行 range query，需要index 那个 column
    - 下面的方案，是从系统的角度来设计： 允许不那么准确的粒度。 不需支持特别复杂的查询。最近21秒 和 最近 20秒 没什么区别。
  + Service：
    - 本身已经很小了，不能再分了
  + storage:
    - 记录了 谁， 在某个时刻，做了什么事情
    - 精确到什么精度：秒
    - A 在第1s login。 如果A在第1秒有超过10次login，需要每次都记录吗？
    - A在第1秒，做了10次，用一个数组表示A[1] = 10。
    - namespace: LOGIN:
      * a[0s] = 1 + 1 + 1…. + 1
    - 在第1分钟，往前60个格子，把所有的value，东西加起来如果超过限制，就返回404.
    - 可以用memcached作为存储结构 --因为 我们不希望这个降低系统运行效率。数据无须持久化，需要一个在内存的高效存储结构，不会因为rate limiter往DB写太多东西。
    - 在这个系统里面，数据没有必要做 持久化
    - memcached （ttl = 60s），60秒后失效，将event （e.g. URL shortener） + feature （10.11.12.13）+ timestamp, 作为memcached的key （谁在）
    - 每隔60s 去 for 一个60个循环
    - memcached 有 getMany 的接口，可以 先生成所有的key，然后只需要一次网络request，就能get from memcached server
    - 不需要绝对精确，需要大概限制一下就好。
  + 如果限制的为天为单位： 则+1 就不是 1s（否则要存86400个 entry）每隔一个小时去加1，所以只需要 24 个 entry 就可以了。只会有24个memcached的get 操作
  + 如果一秒的很前发生过很多cnt，会有误差，也有解决方法。
  + 从系统设计的角度，这道题允许误差
  + 从算法设计的角度，这道题 不许误差
* **Design Data Dog：**
  + Google URL Shortener 的 trend 最近1年的访问率。一段时间 发生的 曲线-- dashboard
  + Aggregation: 每个点表达的不是1s，而表示的是一周的数据点。不需要知道更精确的粒度（2小时，一天，）
    - trend不需要所有的 点，只需要 其中 的一部分。如果所有的点都放上去，会很锯齿。
    - 每个点需要表示以 周围单位的 数据点
  + 同一个数据，要同时 存在 s 的计数器里面，也要存在 min 的 bucket里，以及 day 的 bucket里
  + 从存储上来说，不能用memcached，需要持久化。但是
    - 2017年时，2010年的数据有没有必要按秒来存储。可以模糊存储。2010年可以只存52个 点，只需要知道 一个 trend
      * Retention：对于旧的数据，呢能够进行瘦身。原来存成s的，
  + 如何减少 DB Write： 将最近15s的访问次数Aggregate放到内存里。每15s往 noSQL写一次，延迟可能有15s，但这个 并不是
  + 时间 +　time interval + value（发生了多少次）
  + 数据离得越久，越不需要精确
  + 15s的数据写入到数据库里。
  + 要点：
    - 分级存储 （1分钟，5分钟，1个小时）
    - Aggregate： 数据整合之后，再整体往数据库写。
  + 对老数据进行瘦身