系统设计实例讲解：如何设计 Ticket Master 系统

作者：东邪老师，九章算法版权所有

学员面经题

一个类似 ticket master 的网站。 说某个时间段开放某明星演唱会订票，大概会同时有 500K QPS 的访问量，一共只有 50K 张票。 订票的过程是用户打开订票网页（不用考虑认证等问题），填一个 text box 说要订几张票，然后 click 一个 button 就打开一个 page，那个 page 会不停的 spin 直到系统能够预留那几张票，如果预留成功，用户会有几分钟时间填写用户信息已经完成支付，如果到期未支付，这些票就自动被系统收回了。每张票都是一样的，没有位置信息什么的。

求教怎么 design 这个系统，我一开始看到 500K QPS 就有点慌乱了。

九章讲师-东邪老师解答

订票网站一直都是世界难题。12306 应该比这个还恐怖。

不要被 500k QPS 吓到。500k 也好，5k 也好，500 也好，分析的方法和思路都是一样的。

这个题的关键首先是给出一个可行解（无论任何系统设计题的关键都是如此），一个核心要实现的功能是，票的预留与回收。在设计可行解的时候，可以先将 500k qps 抛之脑后。假设现在只有 10 个用户来买票。优化的事情，放到 Evolve 的那一步。

按照我们的 SNAKE 分析法来：

Scenario 设计些啥:

- 用户提交订票请求
- 客户端等待订票
- 预留票，用户完成支付
- 票过期，回收票
- 限制一场演唱会的票数。

Needs 设计得多牛？

500k QPS，面试官已经给出响应时间——是在用户点击的一瞬间就要完成预定么？不是，可以让用户等个几分钟。也就是说，500K 的请求，可以在若干分钟内完成就好了。因此所谓的 500k QPS，并不是 Average QPS，只是说峰值是 500k QPS。而你要做的事情并不是在 1 秒之内完成 500k 的预定，而是把确认你收到了购票申请就好了。

Application 应用与服务

- ReservationService —— 用户提交一个预定请求，查询自己的预定状态
- TicketService —— 系统帮一个预定完成预定，生成具体的票

Kilobyte 数据如何存储与访问

1. ReservationService —— 用户提交了一个订票申请之后，把一条预定的数据写到数据库里。所以需要一个 Reservation 的 table。大概包含的 columns 有：

id(primary_key)

created_at(timestamp)

concert_id(foreign key)

user_id(foreign key)

tickets_count(int)

status(int)

简单的说就是谁在什么时刻预定了哪个演唱会，预定了几张，当前预定状态是什么（等待，成功，失败）。

2. TicketService —— 系统从数据库中按照顺序选出预定，完成预定，预定成功的，生成对应的 Ticket。表结构如下：

id (primary key)

created_at (timestamp)

user_id (fk)

concert_id (fk)

reservation_id (fk)

status (int) // 是否退票之类的

另外，我们当然还需要一个 Concert 的 table，主要记录总共有多少票：

id (primary key)

title (string)

description (text)

start_at (timestamp)

tickets_amount (int)

remain_tickets_amount (int)

...

总结一下具体的一个 Work Solution 的流程如下：

1. 用户提交一个预定，ReservationService 收到预定，存在数据库里，status=pending

2. 用户提交预定之后，跳转到一个等待订票结果的界面，该界面每隔 5-10 秒钟像服务器发送一个请求查询当前的预定状态

3. TicketService 是一个单独执行的程序，你可以认为是一个死循环，不断检查数据库里是否有 pending 状态的票，取出一批票，比如 1k 张，然后顺利处理，创建对应的 Tickets，修改对应的 Reservation 的 status。

Evolve

分析一下上述的每个操作在 500k qps 的情况下会发生什么，以及该如何解决。

1. 用户提交一个预定，ReservationService 收到预定，存在数据库里，status=pending

也就是说，在一秒钟之内，我们要同时处理 500k 的预定请求，首先 web server 一台肯定搞不定，需要增加到大概 500 台，每台 web server 一秒钟同时处理 1k 的请求还是可以的。数据库如果只有一台的话，也很难承受这样大的请求。并且 SQL 和 NoSQL 这种数据库处理这个问题也会非常吃力。可以选用 Redis 这种既是内存级访问速度，又可以做持久化的 key-value 数据库。并且 Redis 自带一个队列的功能，非常适合我们订票的这个模型。Redis 的存取效率大概是每秒钟几十 k，那么也就是我们要大概 20 台 Redis 应该就可以了。我们可以按照 user_id 作为 shard key，分配到各个 redis 上。

2. 用户提交预定之后，跳转到一个等待订票结果的界面，该界面每隔 5-10 秒钟像服务器发送一个请求查询当前的预定状态

使用了 redis 的队列之后，如何查询一个预定信息是否在队列里呢？方法是 reservation 的基本信息除了放到队列里 还需要同时继续存一份在 redis 里。队列里可以只放 reservation_id。此时 reservation_id 可以用 user_id+concert_id+timestamp 来表示。

3. TicketService 是一个单独执行的程序，你可以认为是一个死循环，不断检查数据库里是否有 pending 状态的票，取出一批票，比如 1k 张，然后顺利处理，创建对应的 Tickets，修改对应的 Reservation 的 status。

为每个 Redis 的数据库后面添加一个 TicketService 的程序（在某台机器上跑着），每个 TicketService 负责一个 Redis 数据库。该程序每次从 Redis 的队列中读出最多 1k 的数据，然后计算一下有需要多少张票，比如 2k，然后访问 Concert 的数据库。问 Concert 要 2k 的票，如果还剩有那么多，那么就 remain_tickets_amount - 2k，如果不够的话，就返回还有多少张票，并把 remain_tickets_acount 清零。这个过程要对数据库进行加锁，可以用数据库自己带的锁，也可以用 zookeeper 之类的分布式锁。因为现在是 1k 为一组进行处理，所以这个过程不会很慢，存 Concert 的数据库也不需要很多，一台就够了。因为就算是 500k 的话，分成 500 组，也就是 500 个 queries 峰值，数据库处理起来绰绰有余额。

假如得到了 2k 张票的额度之后，就顺序处理这 1k 个 reservation，然后对每个 reservation 生成对应的 tickets，并在 redis 中标记 reservation 的状态，这里的话，tickets 的 table 大概就会产生 2k 条的 insert，所以 tickets 的数据库需要大概能够承受 20 x 1k = 20k 的并发写。这个的话，大概 20 台 SQL 数据库也就搞定了。

从头理一下

开放订票，500k 的请求从世界各地涌来

通过 Load Balancer 纷发给 500 台 Web Server 。每台 Web Server 大概一秒钟处理 1k 的请求

Web Server 将 1k 的请求 ,按照 user_id 进行 shard ,丢给对应的 redis 服务器里的队列，并把 Reservation 信息也丢给 Redis 存储。

此时，20 台 Redis，每台 Redis 约收到 25k 的 排队订票记录

每台 Redis 背后对应一个 TicketService 的程序，不断的查看 Redis 里的队列是否有订票记录，如果有的话，一次拿出 1k 个订票记录进行处理，问 Concert 要额度，然后把 1k 的 reservation 对应的创建出 2k 左右的 tickets 出来（假如一个 reservation 有 2 张票平均）。

假如这个部分的处理能力是 1k/s 的话，那么这个过程完成需要 25 秒。也就是说，对于用户来说，最慢大概 25 秒之后，就知道自己有没有订上票了，平均等待时间应该低于 10 秒，因为当 concert 的票卖完了的时候，就无需生成 1-2k 条新的 tickets ,那么这个时候速度会快很多。

存储 tickets 的数据库需要多台，因为需要处理的请求大概是 20k 的 qps，所以大概 20 台左右的 Ticket 数据库。

超时的票回收

增加一个 RecycleService。这个 RecycleService 不断访问 Tickets 的数据库，看看有没有超时的票，如果超时了，那么就回收，并且去 Concert 的数据库里把 remain_tickets_acount 增加。

总结如何攻破 500k QPS 的核心点

核心点就是，500k QPS 我只要做到收，不需要做到处理，那么 500 台 web 服务器+20 台 Redis 就可以了。

处理的的时候，分成 1k 一组进行处理，让用户多等个几秒钟，问题不大。用户等 10 秒钟的话，我们需要的服务器数目就降低 10-20 倍，这是个 tradeoff，需要好好权衡的。

一些可能的疑惑和可以继续进化的地方

问：500 台 Web 服务器很多，而且除了订票的那几秒种，大部分的时候都是闲置浪费的，怎么办？

答：用 AWS 的弹性计算服务，为每场演唱会的火爆指数进行评估，然后预先开好机器，用完之后就可以销毁掉。

问：为什么不直接用 Redis 也来存储所有的数据信息？

答：因为是针对通同一个 Concert 的预定，大家需要访问同一条数据(remain_tickets_acount)，shard 是不管用的，Redis 也承受不住 500k QPS 对同一条数据进行读写，并且还要加锁之类

的保证一致性。所以这个对 remain_tickets_acount 的值进行修改，创建对应的 tickets 的过程，是不能在用户请求的时候，实时完成的，需要延迟进行。

问：redis 又用来做队列，又用来做 Reservation 表的存储，是否有点乱？

答：是的，所以一个更好的办法是，只把 redis 当做队列来用 和 Reservation 信息的 Cache 来用。当一个 Reservation 被处理的时候，再到 SQL 数据库里生成对应的持久化记录。这样的好处是，Redis 这种结构其实不是很擅长做持久化数据的存储，我们一般都还是拿来当队列和 cache 用得比较多。

我用了 2 个小时来写这个帖子回复你，是不是该给我一个小红花！