性能数据分析

100万 日活用户
活动当天：预计2倍的用户访问，也就是200万

秒杀前半个小时登录，QPS = 200万/1800 =1111/s
秒杀开始，假设在30秒内完成，平均每个用户点击5次，则TPS = 200万*5/30 = 35万/s

对于此次秒杀活动，采取如下几点的方案

- 热点数据处理
- 缓存设计
- 削峰方案
- 应用负载均衡
- 存储设计
- 高可用设计

热点数据处理

1、让卖家通过报名参加秒杀的方式提前把秒杀商品筛选出来。得到热点数据，提前处理。如提前进行缓存；将秒杀物品放在一个处理队列中，防止请求过大，影响别的商品业务

2、秒杀系统，商品是选定好的，秒杀的页面单独设计，和原有的页面分开

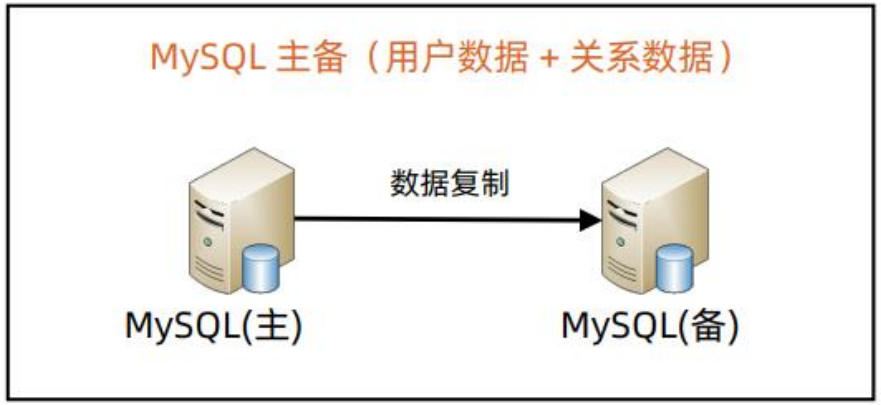
3、对于动静数据，商品详情页等动静资源分离，静态资源可以使用 CDN +Web缓存 进行加速；动态页面的加载可以采用 ESI（Edge Side Includes）或者 CSI（Client Side Include）的方案。

4、让大部分请求和数据直接在 Nginx 服务器或者 Web 代理服务器Varnish上直接返回，Java 层只需处理少量数据的动态请求

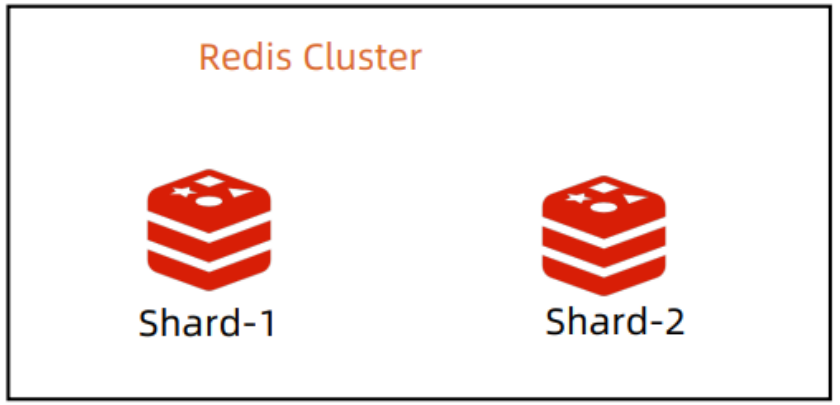
存储架构设计

- MySQL 数据库负责事务。如库存的读写。 采用库存分库分表的方式进行分担
- 订单信息，由于是秒杀，商品数量有限。按照正常情况处理

• 商品信息、订单、支付账单，可先采用 MySQL 主从架构
• 商品详情页等动静资源分离，静态资源可以使用 CDN 进行加速
• 用户信息、库存信息缓存到 Redis, redis 单机 TPS 为 5~10 万， 而在秒杀时有高并发的写，TPS 为 35 万/s，采用多机的 Cluster 架构.

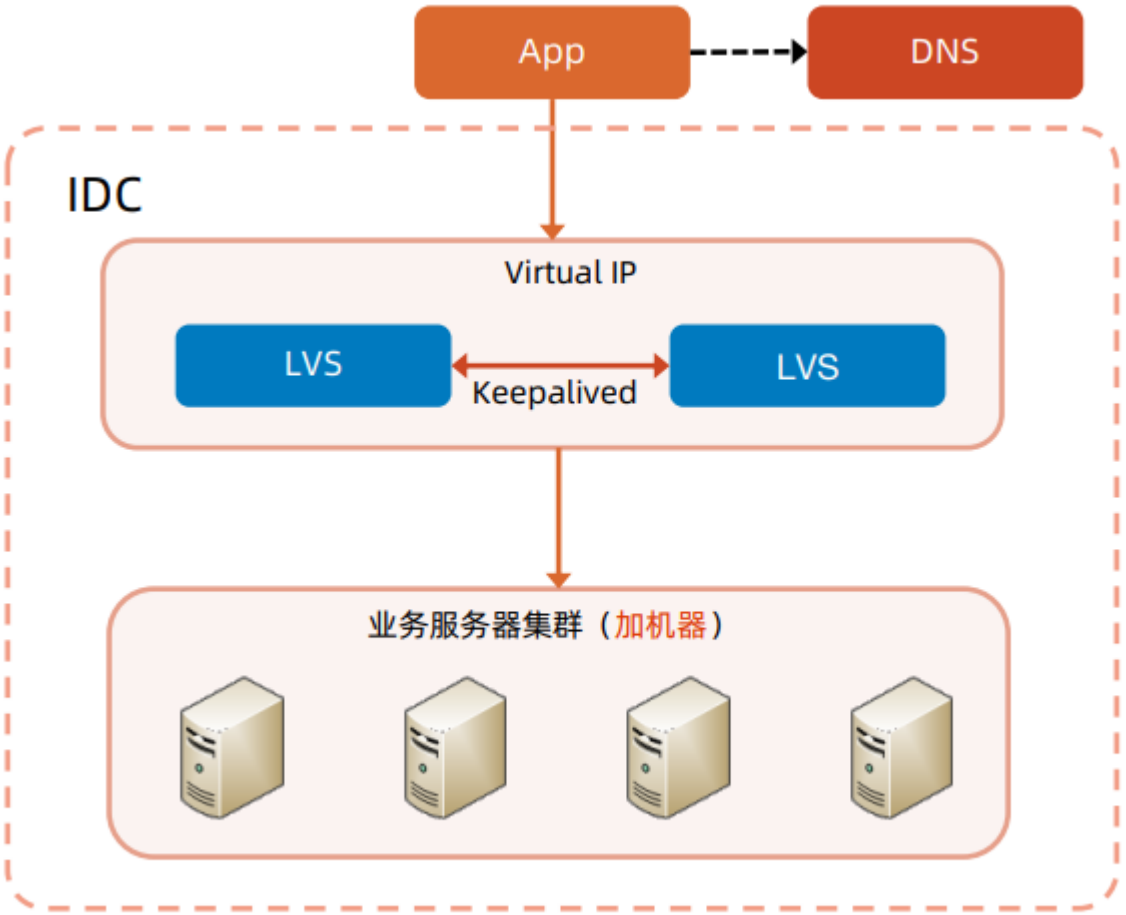


1111/s 的登录用户请求



秒杀信息数据35 万/s

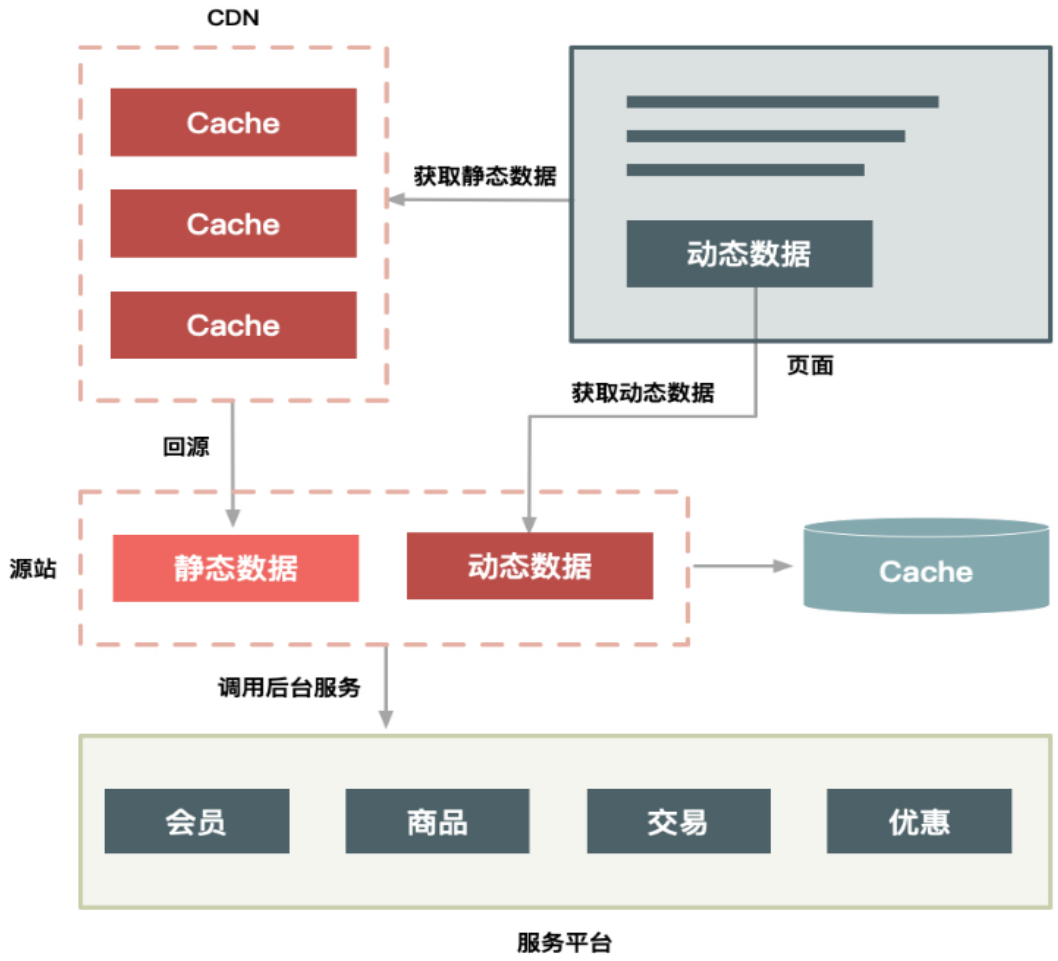
负载均衡设计



活动当天：预计2倍的用户访问，也就是200万
采用LVS 负载均衡，同时让大部分请求和数据直接在Nginx 服务器或者 Web 代理服务器Varnish上直接返回，Java 层只需处理少量数据的动态请求

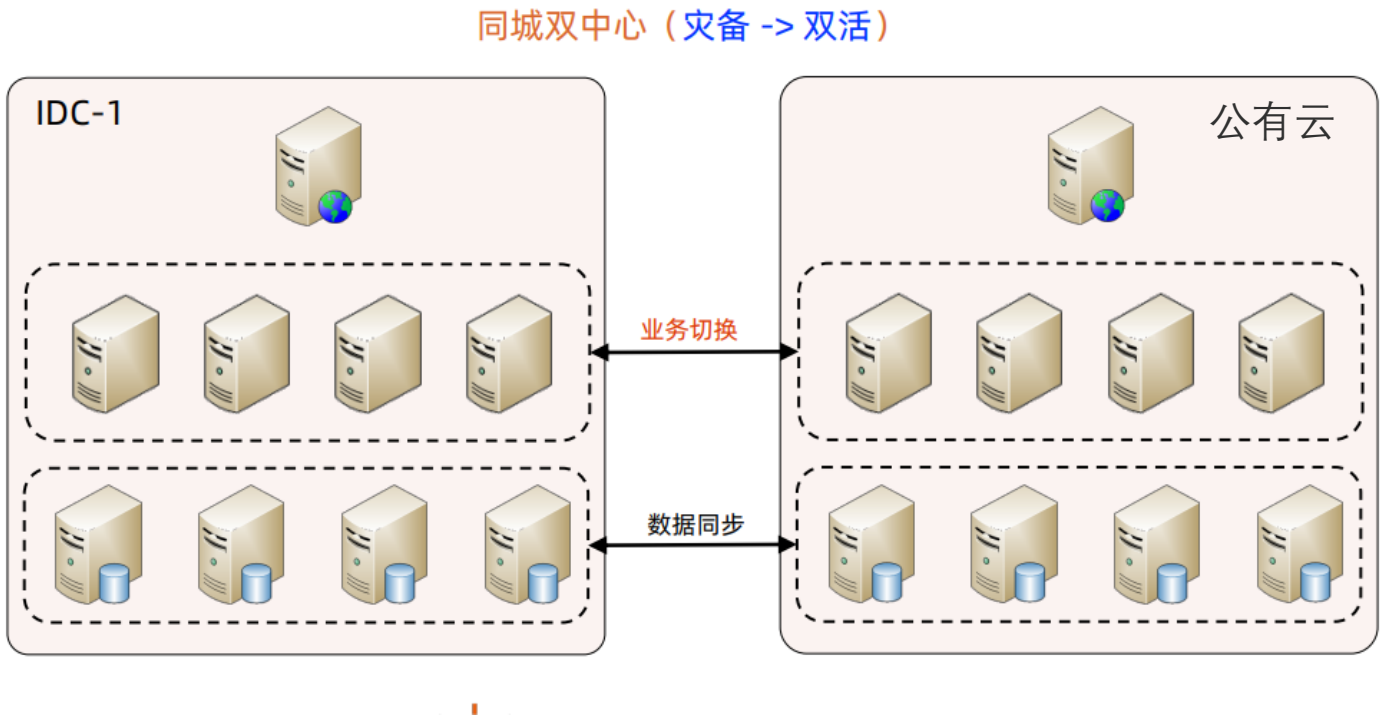
缓存设计

- 采用 web 缓存+CDN 的方式
  选择 CDN 的二级 Cache 比较合适，因为二级 Cache 数量偏少，容量也更大，让用户的请求先回源的 CDN 的二级 Cache 中，如果没命中再回源站获取数据。



高可用设计

目前只有一个机房，且老板要求确保秒杀万无一失，可以优先购买云服务，将需要秒杀应用和数据同步部署到公有云服务上。活动前，搭建好主备数据库同步，同时做好业务的高可用。



削峰方案

1、降级：比如当秒杀流量达到 40w/s 时，把成交记录的获取从展示 20 条降级到只展示 5 条；延长秒杀商品的处理时间

2、限流：

1）使用基于 QPS 和线程数的限流，超过最大 QPS 来进行限流保护；或者在远程调用时设置连接池的线程数，超出这个并发线程请求，就将线程进行排队或者直接超时丢弃。

2）APP 接入端限流：

秒杀活动场景：通过答题器降低接入请求，防止用户高频请求，延长下单时间

网关入口限流：业务规则过滤，检查库存

秒杀任务排队：采用漏桶限流或者先进先出的排队机制，将最后的合法秒杀请求放到限流桶队列，队列满则丢弃

削峰方案

3）交易性写请求场景：分层过滤

假如请求分别经过 CDN、前台读系统（如商品详情系统）、后台系统（如交易系统）和数据库这几层，那么：大部分数据和流量在用户浏览器或者 CDN 上获取，这一层可以拦截大部分数据的读取；经过第二层（即前台系统）时数据（包括强一致性的数据）尽量得走 Cache，过滤一些无效的请求；再到第三层后台系统，主要做数据的二次检验，对系统做好保护和限流，这样数据量和请求就进一步减少；最后在数据层完成数据的强一致性校验。