【年度案例】小米抢购限流峰值系统「大秒」架构解密

原创：马利超 高可用架构 2015-12-04

编者按：高可用架构推出2015年度案例系列文章，分享在架构领域具有典型意义的年度案例，本文根据小米工程师马利超的分享记录。转载请注明高可用架构公众号ArchNotes。

马利超

小米科技的系统研发与大数据工程师，2013年毕业于大连理工大学，毕业后有幸加入小米抢购系统团队，并参与了小米抢购系统开发、重构与调优。其人热爱技术，对分布式系统架构、高并发峰值系统、大数据领域、反作弊领域、搜索/广告/推荐系统有浓厚的兴趣。

上文介绍了【年度案例】小米抢购限流峰值系统架构历年演进历程 ，本文主要介绍最新版「大秒」系统架构。

整合的抢购限流峰值系统——「大秒」

2014年初，公司决定举办一场“米粉节”活动，全天6轮活动，多个国家、多款爆品同时参与抢购。业务场景将变得更加复杂，当天的并发压力也会有一个量级的提升，原有的抢购系统已经不能适应如此复杂的业务场景了。

为此，小米网技术团队基于对 golang 应对高并发、大规模分布式系统能力的肯定，完全基于 golang，重新设计了抢购系统，也就是我们目前使用的抢购限流峰值系统——“大秒”。

在整个系统设计的之初，我们充分考虑了

1. 灵活性及可运营性；
2. 可运维性及可伸缩性；
3. 限流与抢购放号的精准性；

从大秒第一天诞生到演化至今有很多次重构与优化，但一直沿用了设计之初的结构，接下来我们一起了解下小米网抢购限流峰值系统当前的架构以及填过的一些坑。

大秒系统的架构设计

大秒系统主要由如下几个模块构成

1. 限流集群 HTTP 服务
2. 放号策略集群 Middle 服务
3. 监控数据中心 Dcacenter
4. 监控管理体系 Master
5. 准实时防刷模块 antiblack
6. 基础存储与日志队列服务: Redis 集群、Kafka 集群等

整个大秒体系中大秒前端模块 (HTTP/middle/antiblack) 和监控数据中心使用 golang 开发，大秒监控管理体系使用 Python + golang 开发。

大秒的前端架构设计

大秒前端的架构设计从三个系统展开

1. 限流集群 HTTP 服务
2. 策略集群 Middle 服务
3. 准实时反作弊 antiblack 服务

1、限流集群 HTTP 服务

抢购高峰时，通常会有几百万的用户同时请求，瞬时流量非常大，HTTP 集群顶在最前线，接受用户的请求，将合法的请求发送的处理队列，处理队列设置一定的长度限制，通常情况下，抢购用户数与销售商品的比例在100：1，甚至更高，为了避免系统不被冲垮，

保障绝大多数用户的体验，我们认为流量是部分可丢失的，当处理队列满时，丢弃入队请求；

虽然设计上过载流量是部分可丢弃的，但是策略层处理能力是非常 power 的，即便是需要丢弃流量，也是按流量的恶意程度，逐级丢弃的，正常用户购买请求不受影响。

我们使用基于规则的识别、离线画像信息、机器学习逻辑回归等方法，识别恶意用户，在系统高负载的情况下，这部分请求可以优先阻击其发送到策略层，优先处理正常用户的请求，保障用户体验过。

HTTP集群中不同节点之间的所持用的状态数据是一致的，处理逻辑也是一致的，所以整个集群中的任何一个节点挂掉，在前端负载均衡能力下，服务的准确性与一致性不受任何影响。

2、策略集群 Middle 服务

HTTP 模块将满足条件用户的请求按照 uid 哈希的规则，转发到 Middle 集群中相应的节点，Middle 集群根据商品放号策略判断 (uid:sku:time) 组合是否可以分配购买资格，并返回给相应的 HTTP 服务；

使用 Middle 服务本地内存维护用户的购买记录信息，支持各种购买规则，比如：单次活动不限购买数量，单次活动仅限购买一款商品，单次活动每款商品仅限购买一次。

我们将 Middle 的放号逻辑抽象成一个有限状态机，由商品的放号策略配置阈值来触发放号状态转换，整个配置由 Master 节点统一管理与调度。

为了提升整个系统的处理能力，我们将用户状态数据局部化，单用户（uid）的所有相关信息全部路由到一台 Middle 节点上处理。

但是有一点风险是，Middle 集群中服务可能会出现活动过程中挂掉的风险，在抢购场景下，商品基本上是瞬时卖完，为了保障系统的处理能力，我们主要从代码层面做优化，review 代码逻辑，保证服务应对异常的处理能力。

虽然理论上存在风险，但是在实际工程中，经历过几百次活动，还没出现 Middle 节点挂掉的情况。

3、准实时防刷 antiblack 服务

基于日志流的防刷架构，在每台 HTTP 节点上部署日志收集 Agent，使用高吞吐量的 Kafka 做日志转储队列，antiblack 模块实时分析用户请求日志，基于 IP 粒度、Uid 粒度等做防刷。

虽然此处将 antiblack 模块定义为准实时防刷模块，但是作弊信息识别的延迟时长在 1 分钟之内，其中主要的时延发生在日志的转储过程中。

大秒的监控管理体系

1、监控数据中心 dcacenter

监控数据中心数据种类
(1) 业务级数据：过大秒的商品配置数据与实时状态数据，当前活动的配置与状态数据等；

(2) 系统级数据: 大秒前端服务集群通信地址配置，限流队列初始长度配置，系统服务资源占用情况，包括：CPU、MEM、连接数等；

数据采集方式
同时使用push和pull模式采集业务级监控数据和系统级监控数据，业务级数据越实时越好，做到1秒采集处理，3秒可视化；

对于 HTTP 节点和 Middle 节点采用pull的模式拉去系统监控数据和业务监控数据，优点如下
(1) 灵活性高
由数据中心控制监控数据采集的粒度，在数据中心处理能力既定的情况下，可以根据前端集群的伸缩规模，灵活的调整数据采集的粒度，比如米粉节时，大秒前端集群扩容至过百台，管理的过大秒商品的数量在400个左右，业务级监控数据量很大，此时监控数据采集时间间隔很容易降配至 2s。

对于除Http服务和Middle服务之外的服务集群，如：redis，管理平台各个模块等可以使用监控数据采集agent，将采集到的数据周期性的push到redis队列，dcacenter采集协程实时的从redis队列中拉去消息，对于基础服务以及python实现的服务，增加了监控数据采集灵活性。

(2) 增强服务的可靠性与伸缩性
大秒在设计之初采用push的方式，在每台前端机器上部署一个数据采集agent，agent和大秒前端服务同时alive，才代表抢购系统健康运行。这样即增加了系统的不稳定因素，由

不利于系统的伸缩，将监控数据采集逻辑内置到前端golang程序中，提供tcp管理端口，在数据中心使用pull方式采集数据，很好的解决了这个问题。减少了服务的数量，增强了整个系统的可靠性与伸缩性。

数据ETL与数据缓存

dcacenter同时负责将采集到的业务级数据及系统级监控数据，实时清洗，提取，转换，结构化，并将结构化的数据存储在自身内存中，定制通信协议（golang实现类redis通信协议），作为一个数据中心，对整个管理体系Master及其他系统提供实时数据支持。

将dcacenter直接作为数据中心，主要是出于数据的实时性考虑，省去中间转储环节，上层可视化系统、自动化活动控制系统、规则引擎系统等可以第一时间获得前端实时的销售状态数据及服务的状态数据。

2、监控管理中心 Master

监控管理中心的主要模块如下。

a.仓储库存同步服务StockKeeper

同步商品的仓储系统中的实时库存到秒杀系统，大秒系统拥有双库存保障，一个是实时仓储库存，一个是虚拟库存也就是资格号，在抢购场景下只有当两个库存都有货时，才能正常销售。

b.商品策略控制器PolicyKeeper

基于相应的策略触发器（时间区间与库存区间），当策略触发时，比如12点整，抢购开始，为相应的商品配置策略，并向大秒前端广播商品配置变更命令，在通信基础模块的保障下，整个过程秒级内完成。

c.活动自动化控制ActKeeper

基于监控数据中心获取大秒前端的实时销售数据，自动化的控制活动中的各个状态，活动开始前逐层打开开关，活动开始时打开最后开关，活动过程中维护活动的售罄状态，活动结束后初始化，整个抢购活动的过程无需人工介入；

d.数据可视化

从监控数据中心提取实时的结构化系统级监控数据和业务级监控数据，将活动过程中的详细数据实时可视化到管理页面上，让运营与销售以及大秒管理员能够及时了解当前活动状态，并人工干预活动；

e.监控规则引擎

监控规则引擎建立在监控数据中心之上，根据结构化监控数据判断当前整个抢购系统的状态，及时报警，以及半自动化控制。

f.其他

大秒管理端管理大秒前端所有的数据、配置以及状态，Master体系提供了详细的管理工具

与自动化服务。如果清理大秒前端Middle服务中的用户购买信息等。

3、大秒配置管理数据流

整个抢购系统由 Master 体系中各个服务做统一的控制的，Master 控制商品状态及配置数据的变更，控制当前活动的状态，控制商品放号的策略等。

为了保证时效性，商品、活动、系统等配置状态的变更都需要将变更命令广播前端集群，这期间发生了大量的分布式系统间通信，为了保障命令及时下行，我们提取出了命令转发服务：MdwRouter，用于广播控制命令到大秒前端集群。该服务模块维护了到大秒前端长连接，接收 Master 下发的控制命令，并瞬时广播，保障了整个控制流的处理能力。

举个例子，2015 年米粉节，我们单机房大秒集群的规模在过百台级别，假设为 100 台，管理的独立的商品id的数量在 400 个左右，在这种量级的活动下，商品的放行策略是批量管理的，比如我们根据后端交易系统的压力反馈，调整所有商品的放行速度，这时候需要广播的命令条数在: 100*400=40000 级别，Mdwrouter 很好的保障了系统命令下行的速度，秒级完成命令下行。

小米抢购技术架构

1、小米抢购服务闭环设计

小米网抢购系统服务见上图

1. bigtap体系中大秒前端服务负责抢购时限流放号，并控制放号策略以及维护用户在本地缓存中的购买记录。
2. cart服务验证token的有效性，并向counter服务发起销量验证请求;
3. counter服务是整个抢购系统最终的计数器， 海量的请求在bigtap服务的作用下已经被限制在可以承受的压力范围内，并且复杂的放号策略已经在大秒Middle服务中实现，counter只负责最终的计数即可。counter服务采用redis记录相应商品的放号情况，根据预设的销量，判断当前请求加购物车商品是否有库存余量，并维护商品销量;
4. bigtap体系中的dcacenter服务实时采集商品销量，Master中活动自动化控制服务依据商品销量判断当前商品是否售罄，售罄则通过设置商品的售罄状态，并通知大秒前端;

2、2015年米粉节介绍

从上述整个服务闭环设计可以看出，大秒的功能完全可以抽象成限流系统，只有在处理抢购活动时，数据的管理与一致性要求才使整个系统变得复杂。

2015年米粉节，我们完全使用大秒的限流功能，不限用户的购买数量，很便捷的将系统部署在两个机房，一个物理机房，一个公有云集群，两者同时服务，大秒系统作为整个商城的最前端，能够根据后端服务的压力状态，瞬时调整整个集群放行流量大小，非常好的保障了整个米粉节的正常举行。

在上述文章中，已经介绍了一些服务设计的出发点，每一次优化的背后，都至少有一次惨痛的经历。

大秒系统架构的几点经验总结

1、Golang GC 优化方法

我们从 golang 1.2 版本开始在线上抢购系统中大规模使用，最初上线的 TC 限流集群在抢购的过程中通过过载重启的方式瘸腿前行。

在当前的大秒系统中，对于限流集群主要是 goroutine 资源、HTTP 协议数据结构、TCP 连接读写缓冲区等频繁动态开销，造成内存 GC 压力大，在现有 GC 能力下，我们对 GC 优化从以下几个方面考虑

1. 减少垃圾产生：降低数据结构或者缓冲区的开销；
2. 手动管理内存：使用内存池，手动管理内存；
3. 脏数据尽快释放，增大空闲内存比。

我们使用了以下 3 种 golang GC 优化方法

1）定制 golang HTTP 包

调整 HTTP 协议 conn 数据结构默认分配读写缓冲区的大小，以及手动维护读写缓存池，减少动态开辟内存的次数，降低 GC 压力。

在 Go 语言原生的 HTTP 包中会为每个请求默认分配 8KB 的缓冲区，读、写缓冲区各 4K。而在我们的服务场景中只有 GET 请求,服务需要的信息都包含在 HTTP header 中，并没有 body，实际上不需要如此大的内存进行存储，所以我们调小了读写缓冲区，将读缓冲区调小到 1K，写缓冲区调小到 32B，golang 的 bufio 在写缓冲区较小时，会直接写出。

从 golang 1.3 开始，HTTP 原生的包中已经使用了sync.Pool 维护读写缓存池，但是 sync.Pool 中的数据会被自动的回收，同样会小量的增加 GC 压力，我们此处自己维护缓存池来减少垃圾回收。

2）加快资源释放

原生的 HTTP 包默认使用 keep-alive 的方式，小米抢购场景下，恶意流量占用了大量的连接，我们通过主动设置 response header 的 connection 为 close 来主动关闭恶意连接，加快 goroutine 资源的释放。

3）升级版本

跟进使用 golang 最新的版本，golang 后续的每个版本都有针对 GC 能力的调整。

得益于开源技术力量，以及大秒系统在 GC 优化上的努力，以及系统层的调优，我们的 HTTP 限流层已经可以余量前行。

从上图可以看出，得益于 GC 的优化，2015 年米粉节，每轮抢购，HTTP 服务的内存不会有特别大的抖动。

2、HTTP 服务器内存调优之操作系统参数调整

我们的服务场景下绝大多数的请求数都是恶意请求，恶意请求通常都是短连接请求，大量的短连接会处于 timewait 状态，几分钟之后才会释放，这样会占用大量的资源，通过调整内核参数，尽快释放或者重用 timewait 状态的连接，减少资源的开销。

具体参数调整如下：

```
net.ipv4.tcp_tw_recycle = 1 （打开TIME-WAIT sockets快速回收）
net.ipv4.tcp_tw_reuse = 1 （允许TIME-WAIT sockets复用）
net.ipv4.tcp_max_tw_buckets=10000 （降低系统连接数和资源占用，默认为18w）
```

高并发场景下，操作系统层网络模块参数的调整，会起到事半功倍的效果。

3、没有通信就谈不上分布式系统

整个大秒系统模块之间面临的通信要求是非常苛刻的，Master 节点与 HTTP、Middle 节点要频繁的广播控制命令，dcacenter要实时的收集 HTTP、Middle 节点的监控管理数据，HTTP 要将用户的购买请求路由到 Middle 节点之间，Middle 节点要返回给相应的 HTTP 节点放号信息；

我们基于 TCP 定制了简单、高效的通信协议，对于 HTTP 层和 Middle 层通信，通信模块能够合并用户请求，减少通信开销，保障整个大秒系统的高效通信，增加服务的处理能力。

4、服务闭环设计

从上述抢购的服务闭环架构中可以看出，整个抢购流程处理bigtap系统之外，还有 cart 服务，中心 counter 服务，这三者与 bigtap 系统构成了一个数据流的闭环，但是在大秒最初的设计中，是没有 counter 服务的，Middle层策略集群在放号的同时，又作为计数

服务存在，但是整个抢购流程却是以商品加入购物车代表最终的抢购成功，这在设计上有一个漏洞，假如 bigtap 计数了，但是token 并没有请求加购物车成功，这是不合理的。为了保证整个系统的准确性，我们增加了计数器服务，计数操作发生在加购物车下游，bigtap 在从计数中心取出商品实时销量，由此，构成一个服务闭环设计。在提升了系统的准确性，同时也保证了用户体验。

5、技术的选择要可控

我们一开始选择使用 ZooKeeper 存放商品的配置信息，在抢购活动的过程伴随着大量的配置变更操作，ZooKeeper 的 watch 机制不适合用于频繁写的场景，造成消息丢失，大秒前端集群状态与配置不一致。

后来，我们将所有的配置信息存放在 Redis 中，基于通信模块，在发生配置变更时，伴随着一次配置项变更的广播通知，大秒前端根据相应的通知命令，拉取 Redis 中相应的配置信息，变更内存中配置及状态。

大秒的几点设计原则

1. 分治是解决复杂问题的通则；我们从第一代抢购系统演进到当前的大秒系统，衍生出了很多服务，每个服务的产生都是为了专门解决一个问题，分离整个复杂系统，针对每个服务需要解决的问题，各个击破，重点优化。由此，才保障了秒杀体系整体性能、可靠性的提升；

2. 服务化设计；系统解耦，增强系统的伸缩性与可靠性；

3. 无状态设计，增强系统的伸缩性，提升集群整体处理能力；

4. 状态数据局部化，相对于数据中心化，提升集群整体处理能力。

5. 中心化监控管理，热备部署，既保证了服务的高可用性，又能够提升开发和管理效率。随着集群规模的增大以及管理数据的增多，分离管理信息到不同的数据管理节点，实现管理能力的扩容。通常情况下，中小型分布式系统，单机管理能力即可满足。

6. 避免过度设计，过早的优化；小步快跑，频繁迭代。

7. 没有华丽的技术，把细小的点做好，不回避问题，特别是在高并发系统中，一个细小的问题，都可以引发整个服务雪崩。

(generated by haroopad)

Q&A

1、实时仓库怎么避免超卖?

我们的抢购系统以加入购物车代表购买成功，因为用户要买配件等，库存是由计数器控制的，先限流，在计数，在可控的并发量情况下，不会出现超卖。

2、有了放号系统计算放号规则，为什么还需要一个外围的 counter?

主要是 bigtap 到 cart 的环节 token 有丢失，在 cart 之后再加一个计数器，保障销量，bigtap 再读取计数器的数据控制前端商品销售状态，整个延迟不超 3s。

3、HTTP 集群通过 uuid hash 到 Middle，如果目标 Middle 已经死掉怎么应对?

这个问题在文章中有强调，在我们的场景下，商品迅速卖完，这块没有做高可用，只是从代码层面做 review，完善异常处理机制，并且通常情况下，middle 负载不是特别高，几百次活动下来，还没出现过挂掉情况。

4、防刷系统是离线计算的吗，还是有在线识别的策略?

基于日志，准实时，因为请求量比较大，专门搭了一套 Kafka 服务转储日志，基于 golang 开发 logcollect 与 antiblack 模块，可以达到很高的处理性能。

5、请问如何模拟大量请求做测试?

我们遇到的情况是，由于压测机单机端口限制造成早期不好测试，我们这边压测团队基于开源模块开发了能够模拟虚拟IP的模块，打破了单机端口的限制。

6、即使广播和 Redis 拉取商品配置信息，仍有可能配置信息不一致如何解决?

这个主要是商品的配置和状态信息，不涉及到强一致性要求的场景，我们这样可以在秒级达到最终一致性。

想更多了解小米抢购的架构，可阅读抢购系统架构演进介绍 【年度案例】小米抢购限流峰值系统架构历年演进历程

本文策划郭军@360、编辑刘世杰@京东，转载请注明来自"高可用架构(ArchNotes)"微信公众号。高可用架构聚焦互联网架构分享，咨询投稿或报名分享请回复“speaker”。