《程序员》杂志精选：“米粉节”背后的故事——小米网抢购系统开发实践

CSDN 2014-11-10

2014年的米粉节

2014年4月9日凌晨，我和同事们对小米网的抢购系统做了最后的检查与演练。几个小时后，小米网今年开年来最重要的一次大型活动“米粉节”就要开始了。

这次米粉节活动，是小米电商的成人礼，是一次重要的考试。小米网从网站前端、后台系统、仓储物流、售后等各个环节，都将接受一次全面的压力测试。

10点整，一波流量高峰即将到来，几百万用户将准点挤入小米网的服务器。而首先迎接压力冲击的，就是挡在最前面的抢购系统。

而这个抢购系统是重新开发、刚刚上线不久的，这是它第一次接受这样严峻的考验。

系统能不能顶住压力？能不能顺畅正确地执行业务逻辑？这些问题不到抢购高峰那一刻，谁都不能百分百确定。

9点50分，流量已经爬升得很高了；10点整，抢购系统自动开启，购物车中已经顺利加入了抢购商品。

一两分钟后，热门的抢购商品已经售罄自动停止抢购。抢购系统抗住了压力。

我长舒一口气，之前积累的压力都消散了。我坐到角落的沙发里，默默回想抢购系统所经历的那些惊心动魄的故事。这可真是一场很少人有机会经历的探险呢。

抢购系统是怎样诞生的

时间回到2011年底。小米公司在这一年8月16日首次发布了手机，立刻引起了市场轰动。随后，在一天多的时间内预约了30万台。之后的几个月，这30万台小米手机通过排号的方式依次发货，到当年年底全部发完。

然后便是开放购买。最初的开放购买直接在小米的商城系统上进行，但我们那时候完全低估了“抢购”的威力。瞬间爆发的平常几十倍流量迅速淹没了小米网商城服务器，数据库死锁、网页刷新超时，用户购买体验非常差。

市场需求不等人，一周后又要进行下一轮开放抢购。一场风暴就等在前方，而我们只有一周的时间了，整个开发部都承担着巨大的压力。

小米网可以采用的常规优化手段并不太多，增加带宽、服务器、寻找代码中的瓶颈点优化代码。但是，小米公司只是一家刚刚成立一年多的小公司，没有那么多的服务器和带宽。而且，如果代码中有瓶颈点，即使能增加一两倍的服务器和带宽，也一样会被瞬间爆发的几十倍负载所冲垮。而要优化商城的代码，时间上已没有可能。电商网站很复杂，说不定某个不起眼的次要功能，在高负载情况下就会成为瓶颈点拖垮整个网站。

这时开发组面临一个选择，是继续在现有商城上优化，还是单独搞一套抢购系统？我们决定冒险一试，我和几个同事一起突击开发一套独立的抢购系统，希望能够绝境逢生。

摆在我们面前的是一道似乎无解的难题，它要达到的目标如下：

- 只有一周时间，一周内完成设计、开发、测试、上线；
- 失败的代价无法承受，系统必须顺畅运行；
- 抢购结果必须可靠；
- 面对海量用户的并发抢购，商品不能卖超；
- 一个用户只能抢一台手机；
- 用户体验尽量好些。

设计方案就是多个限制条件下求得的解。时间、可靠性、成本，这是我们面临的限制条件。要在那么短的时间内解决难题，必须选择最简单可靠的技术，必须是经过足够验证的技术，解决方案必须是最简单的。

在高并发情况下，影响系统性能的一个关键因素是：数据的一致性要求。在前面所列的目标中，有两项是关于数据一致性的：商品剩余数量、用户是否已经抢购成功。如果要保证严格的数据一致性，那么在集群中需要一个中心服务器来存储和操作这个值。这会造成性能的单点瓶颈。

在分布式系统设计中，有一个CAP原理。“一致性、可用性、分区容忍性”三个要素最多只能同时实现两点，不可能三者兼顾。我们要面对极端的爆发流量负载，分区容忍性和可用性会非常重要，因此决定牺牲数据的强一致性要求。

做出这个重要的决定后，剩下的设计决定就自然而然地产生了：

1. 技术上要选择最可靠的，因为团队用PHP的居多，所以系统使用PHP开发；

2. 抢资格过程要最简化，用户只需点一个抢购按钮，返回结果表示抢购成功或者已经售罄；
3. 对抢购请求的处理尽量简化，将I/O操作控制到最少，减少每个请求的时间；
4. 尽量去除性能单点，将压力分散，整体性能可以线性扩展；
5. 放弃数据强一致性要求，通过异步的方式处理数据。

最后的系统原理见后面的第一版抢购系统原理图（图1）。

图1 第一版抢购系统原理图

系统基本原理：在PHP服务器上，通过一个文件来表示商品是否售罄。如果文件存在即表示已经售罄。PHP程序接收用户抢购请求后，查看用户是否预约以及是否抢购过，然后检查售罄标志文件是否存在。对预约用户，如果未售罄并且用户未抢购成功过，即返回抢购成功的结果，并记录一条日志。日志通过异步的方式传输到中心控制节点，完成记数等操作。

最后，抢购成功用户的列表异步导入商场系统，抢购成功的用户在接下来的几个小时内下单即可。这样，流量高峰完全被抢购系统挡住，商城系统不需要面对高流量。

在这个分布式系统的设计中，对持久化数据的处理是影响性能的重要因素。我们没有选择传统关系型数据库，而是选用了Redis服务器。选用Redis基于下面几个理由。

1. 首先需要保存的数据是典型的Key/Value对形式，每个UID对应一个字符串数据。传统数据库的复杂功能用不上，用KV库正合适。
2. Redis的数据是in-memory的，可以极大提高查询效率。
3. Redis具有足够用的主从复制机制，以及灵活设定的持久化操作配置。这两点正好是我们需要的。

在整个系统中，最频繁的I/O操作，就是PHP对Redis的读写操作。如果处理不好，Redis服务器将成为系统的性能瓶颈。

系统中对Redis的操作包含三种类型的操作：查询是否有预约、是否抢购成功、写入抢购成功状态。为了提升整体的处理能力，可采用读写分离方式。

所有的读操作通过从库完成，所有的写操作只通过控制端一个进程写入主库。

在PHP对Redis服务器的读操作中，需要注意的是连接数的影响。如果PHP是通过短连接访问Redis服务器的，则在高峰时有可能堵塞Redis服务器，造成雪崩效应。这一问题可以通过增加Redis从库的数量来解决。

而对于Redis的写操作，在我们的系统中并没有压力。因为系统是通过异步方式，收集PHP产生的日志，由一个管理端的进程来顺序写入Redis主库。

另一个需要注意的点是Redis的持久化配置。用户的预约信息全部存储在Redis的进程内存中，它向磁盘保存一次，就会造成一次等待。严重的话会导致抢购高峰时系统前端无法响应。因此要尽量避免持久化操作。我们的做法是，所有用于读取的从库完全关闭持久化，一个用于备份的从库打开持久化配置。同时使用日志作为应急恢复的保险措施。

整个系统使用了大约30台服务器，其中包括20台PHP服务器，以及10台Redis服务器。在接下来的抢购中，它顺利地抗住了压力。回想起当时的场景，真是非常的惊心动魄。

第二版抢购系统

经过了两年多的发展，小米网已经越来越成熟。公司准备在2014年4月举办一次盛大的“米粉节”活动。这次持续一整天的购物狂欢节是小米网电商的一次成人礼。商城前端、库存、物流、售后等环节都将经历一次考验。

对于抢购系统来说，最大的不同就是一天要经历多轮抢购冲击，而且有多种不同商品参与抢购。我们之前的抢购系统，是按照一周一次抢购来设计及优化的，根本无法支撑米粉节复杂的活动。而且经过一年多的修修补补，第一版抢购系统积累了很多的问题，正好趁此机会对它进行彻底重构。

第二版系统主要关注系统的灵活性与可运营性（图2）。对于高并发的负载能力，稳定性、准确性这些要求，已经是基础性的最低要求了。我希望将这个系统做得可灵活配置，支持各种商品各种条件组合，并且为将来的扩展打下良好的基础。

图2 第二版系统总体结构图

在这一版中，抢购系统与商城系统依然隔离，两个系统之间通过约定的数据结构交互，信息传递精简。通过抢购系统确定一个用户抢得购买资格后，用户自动在商城系统中将商品

加入购物车。

在之前第一版抢购系统中，我们后来使用Go语言开发了部分模块，积累了一定的经验。因此第二版系统的核心部分，我们决定使用Go语言进行开发。

我们可以让Go程序常驻内存运行，各种配置以及状态信息都可以保存在内存中，减少I/O操作开销。对于商品数量信息，可以在进程内进行操作。不同商品可以分别保存到不同的服务器的Go进程中，以此来分散压力，提升处理速度。

系统服务端主要分为两层架构，即HTTP服务层和业务处理层。HTTP服务层用于维持用户的访问请求，业务处理层则用于进行具体的逻辑判断。两层之间的数据交互通过消息队列来实现。

HTTP服务层主要功能如下：

1. 进行基本的URL正确性校验；
2. 对恶意访问的用户进行过滤，拦截黄牛；
3. 提供用户验证码；
4. 将正常访问用户数据放入相应商品队列中；
5. 等待业务处理层返回的处理结果。

业务处理层主要功能如下：

1. 接收商品队列中的数据；
2. 对用户请求进行处理；
3. 将请求结果放入相应的返回队列中。

用户的抢购请求通过消息队列，依次进入业务处理层的Go进程里，然后顺序地处理请求，将抢购结果返回给前面的HTTP服务层。

商品剩余数量等信息，根据商品编号分别保存在业务层特定的服务器进程中。我们选择保证商品数据的一致性，放弃了数据的分区容忍性。

这两个模块用于抢购过程中的请求处理，系统中还有相应的策略控制模块，以及防刷和系统管理模块等（图3）。

图3 第二版系统详细结构图

在第二版抢购系统的开发过程中，我们遇到了HTTP层Go程序内存消耗过多的问题。

由于HTTP层主要用于维持住用户的访问请求，每个请求中的数据都会占用一定的内存空间，当大量的用户进行访问时就会导致内存使用量不断上涨。当内存占用量达到一定程度（50%）时，Go中的GC机制会越来越慢，但仍然会有大量的用户进行访问，导致出现“雪崩”效应，内存不断上涨，最终机器内存的使用率会达到90%以上甚至99%，导致服务不可用。

在Go语言原生的HTTP包中会为每个请求分配8KB的内存，用于读缓存和写缓存。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。

为了避免读写缓存的频繁申请和销毁，HTTP包建立了一个缓存池，但其长度只有4，因此在大量连接创建时，会大量申请内存，创建新对象。而当大量连接释放时，又会导致很多对象内存无法回收到缓存池，增加了GC的压力。

HTTP协议是构建在TCP协议之上的，Go的原生HTTP模块中是没有提供直接的接口关闭底层TCP连接的，而HTTP 1.1中对连接状态默认使用keep-alive方式。这样，在客户端多次请求服务端时，可以复用一个TCP连接，避免频繁建立和断开连接，导致服务端一直等待读取下一个请求而不释放连接。但同样在我们的服务场景中不存在TCP连接复用的需求。当一个用户完成一个请求后，希望能够尽快关闭连接。keep-alive方式导致已完成处理的用户连接不能尽快关闭，连接无法释放，导致连接数不断增加，对服务端的内存和带宽都有影响。

通过上面的分析，我们的解决办法如下。

1. 在无法优化Go语言中GC机制时，要避免“雪崩效应”就要尽量避免服务占用的内存超过限制（50%），在处于这个限制内时，GC可以有效进行。可通过增加服务器的方式来分散内存压力，并尽力优化服务占用的内存大小。同时Go 1.3也对其GC做了一定优化。
2. 我们为抢购这个特定服务场景定制了新的HTTP包，将TCP连接读缓存大小改为1KB。
3. 在定制的HTTP包中，将缓存池的大小改为100万，避免读写缓存的频繁申请和销毁。
4. 当每个请求处理完成后，通过设置Response的Header中Connection为close来主动关闭连接。

通过这样的改进，我们的HTTP前端服务器最大稳定连接数可以超过一百万。

第二版抢购系统顺利完成了米粉节的考验。

总结

技术方案需要依托具体的问题而存在。脱离了应用场景，无论多么酷炫的技术都失去了价值。抢购系统面临的现实问题复杂多变，我们也依然在不断地摸索改进。

作者韩祝鹏，小米公司程序员。早期负责MIUI系统发布与运营，后带领小米网系统组设计与开发小米网抢购系统。

本文为《程序员》原创文章，点击“阅读原文”可查看全文并参与讨论。

如果您喜欢这篇文章，请点击右上角“...”将本文分享给你的朋友。

阅读原文