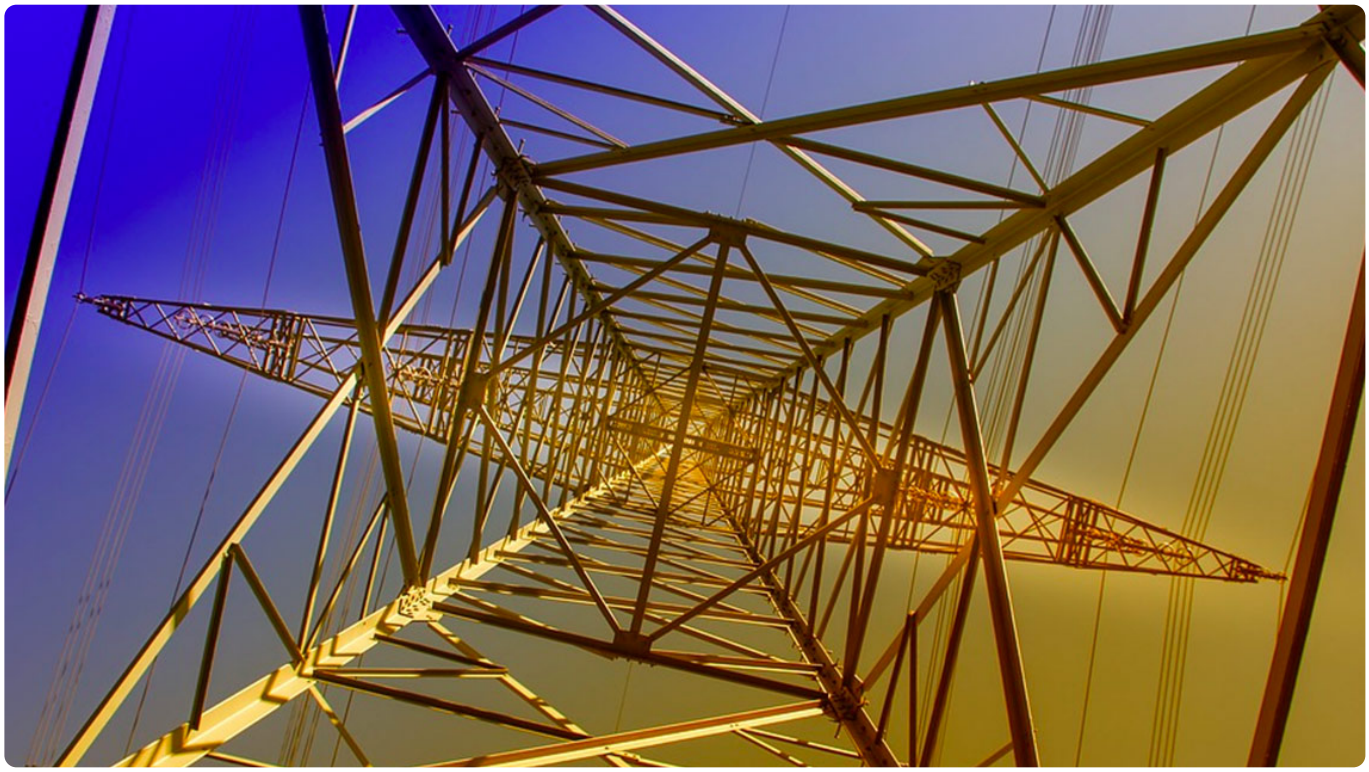
05 | 影响性能的因素有哪些？又该如何提高系统的性能？

2018-10-05 许令波



讲述：秭明
时长 13:28 大小 6.17M

不知不觉，我们已经讲到第五篇了，不知道听到这里，你对于秒杀系统的构建有没有形成一些框架性的认识，这里我再带你简单回忆下前面的主线。

前面的四篇文章里，我介绍的内容多少都和优化有关：第一篇介绍了一些指导原则；第二篇和第三篇从动静分离和热点数据两个维度，介绍了如何有针对性地对数据进行区分和优化处理；第四篇介绍了在保证实现基本业务功能的前提下，尽量减少和过滤一些无效请求的思路。

这几篇文章既是在讲根据指导原则实现的具体案例，也是在讲如何实现能够让整个系统更“快”。我想说的是，优化本身有很多手段，也是一个复杂的系统工程。今天，我就来结合秒杀这一场景，重点给你介绍下服务端的一些优化技巧。

影响性能的因素

你想要提升性能，首先肯定要知道哪些因素对于系统性能的影响最大，然后再针对这些具体的因素想办法做优化，是不是这个逻辑？

那么，哪些因素对性能有影响呢？在回答这个问题之前，我们先定义一下“性能”，服务设备不同对性能的定义也是不一样的，例如 CPU 主要看主频、磁盘主要看 IOPS（Input/Output Operations Per Second，即每秒进行读写操作的次数）。

而今天我们讨论的主要是系统服务端性能，一般用 QPS（Query Per Second，每秒请求数）来衡量，还有一个影响和 QPS 也息息相关，那就是响应时间（Response Time，RT），它可以理解为服务器处理响应的耗时。

正常情况下响应时间（RT）越短，一秒钟处理的请求数（QPS）自然也就会越多，这在单线程处理的情况下看起来是线性的关系，即我们只要把每个请求的响应时间降到最低，那么性能就会最高。

但是你可能想到响应时间总有一个极限，不可能无限下降，所以又出现了另外一个维度，即通过多线程，来处理请求。这样理论上就变成了“总 QPS =（1000ms / 响应时间）× 线程数量”，这样性能就和两个因素相关了，一个是一次响应的服务端耗时，一个是处理请求的线程数。

接下来，我们一起看看这个两个因素到底会造成什么样的影响。

首先，我们先来看看响应时间和 QPS 有啥关系。

对于大部分的 Web 系统而言，响应时间一般都是由 CPU 执行时间和线程等待时间（比如 RPC、IO 等待、Sleep、Wait 等）组成，即服务器在处理一个请求时，一部分是 CPU 本身在做运算，还有一部分是在各种等待。

理解了服务器处理请求的逻辑，估计你会说为什么我们不去减少这种等待时间。很遗憾，根据我们实际的测试发现，减少线程等待时间对提升性能的影响没有我们想象得那么大，它并不是线性的提升关系，这点在很多代理服务器（Proxy）上可以做验证。

如果代理服务器本身没有 CPU 消耗，我们在每次给代理服务器代理的请求加个延时，即增加响应时间，但是这对代理服务器本身的吞吐量并没有多大的影响，因为代理服务器本身的

资源并没有被消耗，可以通过增加代理服务器的处理线程数，来弥补响应时间对代理服务器的 QPS 的影响。

其实，真正对性能有影响的是 CPU 的执行时间。这也很好理解，因为 CPU 的执行真正消耗了服务器的资源。经过实际的测试，如果减少 CPU 一半的执行时间，就可以增加一倍的 QPS。

也就是说，我们应该致力于减少 CPU 的执行时间。

其次，我们再来看看线程数对 QPS 的影响。

单看“总 QPS”的计算公式，你会觉得线程数越多 QPS 也就会越高，但这会一直正确吗？显然不是，线程数不是越多越好，因为线程本身也消耗资源，也受到其他因素的制约。例如，线程越多系统的线程切换成本就会越高，而且每个线程也都会耗费一定内存。

那么，设置什么样的线程数最合理呢？其实**很多多线程的场景都有一个默认配置，即“线程数 = 2 * CPU 核数 + 1”**。除去这个配置，还有一个根据最佳实践得出来的公式：

> 线程数 = [(线程等待时间 + 线程 CPU 时间) / 线程 CPU 时间] × CPU 数量

当然，最好的办法是通过性能测试来发现最佳的线程数。

换句话说，要提升性能我们就要减少 CPU 的执行时间，另外就是要设置一个合理的并发线程数，通过这两方面来显著提升服务器的性能。

现在，你知道了如何来快速提升性能，那接下来你估计会问，我应该怎么发现系统哪里最消耗 CPU 资源呢？

如何发现瓶颈

就服务器而言，会出现瓶颈的地方有很多，例如 CPU、内存、磁盘以及网络等都可能会导致瓶颈。此外，不同的系统对瓶颈的关注度也不一样，例如对缓存系统而言，制约它的是内存，而对存储型系统来说 I/O 更容易是瓶颈。

这个专栏中，我们定位的场景是秒杀，它的瓶颈更多地发生在 CPU 上。

那么，如何发现 CPU 的瓶颈呢？其实有很多 CPU 诊断工具可以发现 CPU 的消耗，最常用的就是 JProfiler 和 Yourkit 这两个工具，它们可以列出整个请求中每个函数的 CPU 执行时间，可以发现哪个函数消耗的 CPU 时间最多，以便你有针对性地做优化。

当然还有一些办法也可以近似地统计 CPU 的耗时，例如通过 jstack 定时地打印调用栈，如果某些函数调用频繁或者耗时较多，那么那些函数就会多次出现在系统调用栈里，这样相当于采样的方式也能够发现耗时较多的函数。

虽说秒杀系统的瓶颈大部分在 CPU，但这并不表示其他方面就一定不出现瓶颈。例如，如果海量请求涌过来，你的页面又比较大，那么网络就有可能出现瓶颈。

怎样简单地判断 CPU 是不是瓶颈呢？一个办法就是看当 QPS 达到极限时，你的服务器的 CPU 使用率是不是超过了 95%，如果没有超过，那么表示 CPU 还有提升的空间，要么是有锁限制，要么是有过多的本地 I/O 等待发生。

现在你知道了优化哪些因素，又发现了瓶颈，那么接下来就要关注如何优化了。

如何优化系统

对 Java 系统来说，可以优化的地方很多，这里我重点说一下比较有效的几种手段，供你参考，它们是：减少编码、减少序列化、Java 极致优化、并发读优化。接下来，我们分别来看一下。

1. 减少编码

Java 的编码运行比较慢，这是 Java 的一大硬伤。在很多场景下，只要涉及字符串的操作（如输入输出操作、I/O 操作）都比较耗 CPU 资源，不管它是磁盘 I/O 还是网络 I/O，因为都需要将字符转换成字节，而这个转换必须编码。

每个字符的编码都需要查表，而这种查表的操作非常耗资源，所以减少字符到字节或者相反的转换、减少字符编码会非常有成效。减少编码就可以大大提升性能。

那么如何才能减少编码呢？例如，网页输出是可以直接进行流输出的，即用 resp.getOutputStream() 函数写数据，把一些静态的数据提前转化成字节，等到真正往外写的时候再直接用 OutputStream() 函数写，就可以减少静态数据的编码转换。

我在《深入分析 Java Web 技术内幕》一书中介绍的“Velocity 优化实践”一章的内容，就是基于把静态的字符串提前编码成字节并缓存，然后直接输出字节内容到页面，从而大大减少编码的性能消耗的，网页输出的性能比没有提前进行字符到字节转换时提升了 30% 左右。

2. 减少序列化

序列化也是 Java 性能的一大天敌，减少 Java 中的序列化操作也能大大提升性能。又因为序列化往往是和编码同时发生的，所以减少序列化也就减少了编码。

序列化大部分是在 RPC 中发生的，因此避免或者减少 RPC 就可以减少序列化，当然当前的序列化协议也已经做了很多优化来提升性能。有一种新的方案，就是可以将多个关联性比较强的应用进行“合并部署”，而减少不同应用之间的 RPC 也可以减少序列化的消耗。

所谓“合并部署”，就是把两个原本在不同机器上的不同应用合并部署到一台机器上，当然不仅仅是部署在一台机器上，还要在同一个 Tomcat 容器中，且不能走本机的 Socket，这样才能避免序列化的产生。

另外针对秒杀场景，我们还可以做得更极致一些，接下来我们来看第 3 点：Java 极致优化。

3. Java 极致优化

Java 和通用的 Web 服务器（如 Nginx 或 Apache 服务器）相比，在处理大并发的 HTTP 请求时要弱一点，所以一般我们都会对大流量的 Web 系统做静态化改造，让大部分请求和数据直接在 Nginx 服务器或者 Web 代理服务器（如 Varnish、Squid 等）上直接返回（这样可以减少数据的序列化与反序列化），而 Java 层只需处理少量数据的动态请求。针对这些请求，我们可以使用以下手段进行优化：

直接使用 Servlet 处理请求。避免使用传统的 MVC 框架，这样可以绕过一大堆复杂且用处不大的处理逻辑，节省 1ms 时间（具体取决于你对 MVC 框架的依赖程度）。

直接输出流数据。使用 resp.getOutputStream() 而不是 resp.getWriter() 函数，可以省掉一些不变字符数据的编码，从而提升性能；数据输出时推荐使用 JSON 而不是模板引擎（一般都是解释执行）来输出页面。

4. 并发读优化

也许有读者会觉得这个问题很容易解决，无非就是放到 Tair 缓存里面。集中式缓存为了保证命中率一般都会采用一致性 Hash，所以同一个 key 会落到同一台机器上。虽然单台缓存机器也能支撑 30w/s 的请求，但还是远不足以应对像“大秒”这种级别的热点商品。那么，该如何彻底解决单点的瓶颈呢？

答案是采用应用层的 LocalCache，即在秒杀系统的单机上缓存商品相关的数据。

那么，又如何缓存（Cache）数据呢？你需要划分成动态数据和静态数据分别进行处理：

- 像商品中的“标题”和“描述”这些本身不变的数据，会在秒杀开始之前全量推送到秒杀机器上，并一直缓存到秒杀结束；
- 像库存这类动态数据，会采用“被动失效”的方式缓存一定时间（一般是数秒），失效后再去缓存拉取最新的数据。

你可能还会有疑问：像库存这种频繁更新的数据，一旦数据不一致，会不会导致超卖？

这就要用到前面介绍的读数据的分层校验原则了，读的场景可以允许一定的脏数据，因为这里的误判只会导致少量原本无库存的下单请求被误认为有库存，可以等到真正写数据时再保证最终的一致性，通过在数据的高可用性和一致性之间的平衡，来解决高并发的数据读取问题。

总结一下

性能优化的过程首先要从发现短板开始，除了我今天介绍的一些优化措施外，你还可以在减少数据、数据分级（动静分离），以及减少中间环节、增加预处理等这些环节上做优化。

首先是“发现短板”，比如考虑以下因素的一些限制：光速（光速：C = 30 万千米 / 秒；光纤：V = C/1.5=20 万千米 / 秒，即数据传输是有物理距离的限制的）、网速（2017 年 11 月知名测速网站 Ookla 发布报告，全国平均上网带宽达到 61.24 Mbps，千兆带宽下 10KB 数据的极限 QPS 为 1.25 万 QPS=1000Mbps/8/10KB）、网络结构（交换机 / 网卡的限制）、TCP/IP、虚拟机（内存 /CPU/IO 等资源的限制）和应用本身的一些瓶颈等。

其次是减少数据。事实上，有两个地方特别影响性能，一是服务端在处理数据时不可避免地存在字符到字节的相互转化，二是 HTTP 请求时要做 Gzip 压缩，还有网络传输的耗时，这些都和数据大小密切相关。

再次，就是数据分级，也就是要保证首屏为先、重要信息为先，次要信息则异步加载，以这种方式提升用户获取数据的体验。

最后就是要减少中间环节，减少字符到字节的转换，增加预处理（提前做字符到字节的转换）去掉不需要的操作。

此外，要做好优化，你还需要做好应用基线，比如性能基线（何时性能突然下降）、成本基线（去年双 11 用了多少台机器）、链路基线（我们的系统发生了哪些变化），你可以通过这些基线持续关注系统的性能，做到在代码上提升编码质量，在业务上改掉不合理的调用，在架构和调用链路上不断的改进。

最后，欢迎你在留言区和我交流，你也可以说说在实际工作中，□关于性能提升还有哪些更好的思路或者方案，我们一起沟通探讨。

上一篇　04 | 流量削峰这事应该怎么做？

下一篇　06 | 秒杀系统“减库存”设计的核心逻辑

精选留言 (22)

写留言

公号-代码...
2018-10-05

13

除了本文提供的方式外，还可从考虑从以下方面进行调整：
1 提升硬件条件：CPU核数、主频、内存、磁盘I/O、SSD、网卡等
2 JVM性能调优
3 缓存

展开

作者回复: 👍

godtrue
2018-11-13

4

很棒，如醍醐灌顶！

性能优化的核心就一个字-减

如果还继续减的...

展开

作者回复: 👍

speedfirst
2018-10-07

3

能否再具体解释下“合并部署”如何避免序列化的？我的理解是不管在不在一个tomcat都要走一次http，所以总要序列化。tomcat提供某种机制可以跨进程非序列化通信？

作者回复: 这个里面实现比较复杂，几句话很难讲清楚，《架构演进与性能优化》有专门一章介绍了实现方案

Sven
2018-12-22

2

我也刚发现原来大神是java技术内幕作者本尊>o<

展开

One day
2018-11-01

2

想问下有dubbo相关的书籍推荐一下吗？在网上没找到。。。

作者回复: Dubbo专门的书籍我也没看到过，不过可以去看看他的官方文档，再结合他的源码，相信你能够搞明白😉

linx
2019-02-18

1

千兆带宽下 10KB 数据的极限 QPS 为 1.25 万 QPS=1000Mbps/8/10KB
不太明白这个8 是指什么呢？

展开

作者回复: 大B和小b的区别

Eliefly
2018-12-17

1

自己道行浅，看这有点泛啊..

展开

作者回复: 还是要自己去实践:)

Hana

2018-10-23

拜读过您的 《深入分析 Java Web 技术内幕》 这一本书，讲解非常通俗易懂，也不失深度，今天才反应过来作者跟您是同一个人😉

展开

作者回复: 😉

看不到de颜...

2018-10-15

1

看完这章内容有一点跟之前了解到不一致的地方。就是通常设置线程数大小会根据任务类型进行区分。通常会划分为IO密集型和CPU密集型。印象中CPU密集型的任务通常线程数建议就是 CPU核心数+1。那么老师在文中提到的 2*CPU核心数+1指的是IO密集型任务的通常线程数设置嘛？
看完这章内容还有一个感受就是设置JSF(类似于Dubbo的一个RPC框架)线程池大小时是...

展开

作者回复: 关于线程数的设定我给出的是一个经验公式，不是所有的线程池都要根据这个设置，那肯定也不合理，例如，一个Java系统里可能都很多线程池，如果每个线程池都设置这个值，那会有很多的线程。

我说的这个设置，其实主要是想把最重要处理用户请求的线程池设置这个数，或者是系统中最核心的处理用户请求的线程池建议按照这个来设置，另外像Dubbo中的线程池的数量还要根据每个请求的rt以及并发数来综合考量，例如如果每个请求的rt比较长，那么并发数一多的话，很容易就满了，这时你为了提升并发请求数，肯定要多设置一些线程数，否则很容易请求失败。当然你也可以增加机器来解决，总之你还是要有个平衡。

吴浩

2018-10-08

1

读的场景可以允许一定的脏数据，导致少量原本无库存的下单请求被误以为有库存，可以在写数据的时候再保证最终一致性

...

展开

作者回复: 嗯，写的一致性主要是通过数据库来保证

SpoutAndBo... 1
2018-10-06

如果让前端无效请求数降低 能否可以使用Nginx的配置 limit_connect来控制

作者回复: 可以到是可以，就是比答题的方式更暴力一点，呵呵

Schelor 1
2018-10-05

许老师行文流畅，文章有层次。
部分文字如统一接入层，Tair等还是可以看出，阿里技术还是渗透很深的。

展开

作者回复: 😉

wuhulala 1
2018-10-05

有个疑问：比如dubbo默认线程池大小是200 这个线程配置 其实在我们的机器往往是8c的 并且是计算密集型 那么就过于大了吧 默认配置这么大是因为大部分机器都是64c+么？

作者回复: Dubbo的线程池的大小还要看看你的远程调用的rt是多少，如果rt比较长，那线程数就要多一点，不然你的系统连接很容易就满了，就拒绝服务了。

Geek_e2b84...
2019-05-07

利用数据库锁的来保证强一致性性能瓶颈还是比较明显的吧，根据商品来做分表对其它业务还是会产品很多影响的，是否可以使用redis缓存来保证数据的一致性，将商品数量缓存到redis，通过redis的串行事务的特性来保证不超卖，毕竟redis的性能还是很强悍的。。。

展开

作者回复: 用Redis来缓存库存也是一种方案，我在文章里也介绍过，不过也有风险，数据持久化会是一个麻烦

ailei
2019-03-31

这个老师比讲linux的刘老师响应留言多，超赞👍

展开

皮卡皮卡丘
2019-02-24

要先换算成字节

展开

这菜真香呀
2019-02-04

最后有点泛泛，性能基线和链路基线具体指什么呢

展开

作者回复: 性能基线和链路基线还可以扩展很多知识点，但是由于篇幅……
简单说来就是性能基线可以以一个系统的关键接口（访问商品）以QPS、RT等指标建立持续的性能测试数据，监控其性能的变化，一旦发现性能下降时可以追溯到什么时间，已经分析出什么原因。

链路基线是以系统的依赖为指标建立的，系统依赖哪些关键接口，要长期加以记录和跟踪。

飞天小侠
2018-12-09

老师，想问下，就是比如在下单的时候，先是写入一个订单，然后再减商品库存，因为减库存是写操作，所以会锁住，那么后面进来的有效请求就会等到释放锁才能操作，这样会不会影响后面的正常请求，响应太慢，然后有没有好的解决方案呢？

展开

作者回复: 锁肯定是会存在的，解决的办法就是尽可能避免产生锁，比如根据商品ID进行分库分表设计；再有就是减少锁的粒度例如阿里对MySQL做了定制优化，可以提升MySQL的并发度

Derek.c

2018-10-19

许老师你好，最近在学习测试Web系统的性能，一般用到的工具wrk、siege、apache ab这些，当测试某一Web应用接口时，以ab举例，是从外网测试(ab -c 10 -n 100 http://a.web.com/)还是从内网测试(ab -c 10 -n 100 http://localhost:80/)，我个人是倾向外网测试（负载均衡、DNS响应速度都要测试）

展开

作者回复: 外网测试和内网测试的主要差别是经过网络的节点的差异，如果你想测试网络带宽对系统的影响可用再外网测试，但是要考虑到对其他业务的影响，例如别把网络带宽打满而影响了其他业务正常服务。

如果仅仅是测试应用程序本身的性能，那么本机测试就足够了。

李俊辉

2018-10-09

如果发现QPS和cpu使用率都没上去怎么办？springboot dubbox zookeeper

作者回复: 那可能什么地方有锁，或者其他的瓶颈例如io