加餐2 ｜答疑精选：这些问题你都清楚吗？

2020-06-08 陶辉

系统性能调优必知必会

进入课程 >

讲述：陶辉

时长 12:13　大小 11.19M

你好，我是陶辉。今天是期中周的第 2 篇加餐，按照约定，这节课我从 1～15 课的留言区精选出了 15 个问题，这里一部分是与内容强相关的，还有一部分是属于拓展型的问题，选择标准就是是否存在增量信息以及问题价值，希望你能从别人的疑问中进行一次自检，引发更多的思考。

第 1 课

鲤鲤鱼：我们集群有一个问题，某一台物理机的 CPU 会被 Hadoop yarn 的查询任务打满，并且占用最多的 pid 在不停的变化，我查看了 TIME_WAIT 的个数好像也不是很多，在顶峰的时候还没达到一万，能够持续一两个小时。这个问题您有没有什么思路呢？

作者：解决性能问题，一般有两种方法：经验派和“理论”派。前者就是基于自己的经验概率，将能想到的优化方法都试一遍，这种方式通常又有效又快速，但无法解决复杂的问题。而所谓理论派，就是沿着固定的思路，使用二分法，从高至低慢慢下沉到细节。

具体到你的问题，我建议你先看看，CPU 占用的是用户态还是系统态，用户态的话就要分析代码了，系统态还要进一步分析。火焰图通常是个很好的办法，虽然搭能画火焰图的环境很麻烦，但这种底层方法很有效（第 19 课会具体讲到火焰图的用法）。

第 2 课

alan：老师好，这节课真好，第一次了解到内存池也是有层次的。我遇到一个问题想请教一下：我有一个和数据库交互的 Groovy 程序，运行起来后会占用很大内存，启动时，将 Xmx 设置为多少，该程序的内存占用就不会超过 Xmx 指定的上限。比如，Xmx=10g，程序就稳定占 10g 内存，但如果不限制的话，最高见过占用 30G 左右。这个您觉得有什么可能的原因吗？

作者：每种 Java 虚拟机都有自己独特的垃圾回收机制，有时为了时间更快就会牺牲更多的内存空间，这是正常的。我建议在服务器上长时间运行的 Java 进程，一定要通过 Xmx 去明确内存占用，否则内存不可控会很麻烦。

第 3 课

杨文宇：链表的内存地址不连续是如何影响序列化的？老师能具体说一下吗？

作者：当数组外还有链表中的元素时，序列化就必须遍历所有元素，比如，至少要做 1 次循环，把每 1 个遍历到的元素的值，序列化写入至另一段内存中。而使用闭散列时，可以直接将这个数组占用的内存作为序列化后的数据。

第 4 课

helloworld：“第二，读取磁盘数据时，需要先找到数据所在的位置，对于机械磁盘来说，就是旋转磁头到数据所在的扇区，再开始顺序读取数据。其中，旋转磁头耗时很长，为了降低它的影响，PageCache 使用了预读功能。”那是不是使用 SSD 这类固态硬盘（不用旋转磁头），PageCache 就没有很大的影响？

作者：对的！其实，当下的操作系统对 SSD 磁盘的支持还不够，当 SSD 广泛应用时，文件系统还需要跟上，得获得很大的性能提升才可以。

第 5 课

Robust：“然而，共享地址空间虽然可以方便地共享对象，但这也导致一个问题，那就是任何一个线程出错时，进程中的所有线程会跟着一起崩溃。”这里的出错应该表示一些特殊的错误吧，或者是说和共享内存有关的错误，比如申请不到内存等。老师，我理解得没错吧？

作者：这里指无法恢复的错误，不仅是内存申请错误，比如访问已经释放的资源，且没有捕获异常或者无法捕获异常，进而操作系统只能杀死线程时，进程里的其它线程也会被杀死。

第 6 课

范闲：用户态的协程不能用互斥或者自旋，会进入内核态与其设计初衷相悖，Python 里面用的 yield。

作者：是的，用户态协程需要用户态的代码将锁重新实现一遍，其中实现时不能用到内核提供的系统调用。

第 7 课

重返归途：广播功能属于双工吗？但多个客户机向主机响应时，会有性能瓶颈吗？

作者：广播不是双工，因为广播是由网络设备实现的，所以服务器无法感知到每个客户端的响应，因此客户端对服务器的响应，与本次广播消息链路无关，它必须是另一个通道。

第 8 课

龙龙：“因此，哪怕有 1 千万并发连接，也能保证 1 万 RPS 的处理能力，这就是 epoll 能在 C10M 下实现高吞吐量的原因。”老师，这段话我不太理解，1 千万的并发连接，只有 1 万的 RPS 这能算高吞吐量吗？相当于每秒只有 1000 个人中的 1 个人得到响应。还是我理解错了，您表述的是另一层意思？

作者：这里有 2 层意思，都是服务于 epoll 的设计思想这一个目的。

1. 这段话的上下文，是指单次获取网络事件时，你可以理解为调用 epoll_wait 系统调用，它的速度与并发连接总数无关，相对于之前的 select/poll 系统调用（它们都与并发连接总数相关），因此 epoll_wait 速度很快，这是实现高吞吐量的关键。
2. 有些应用会长时间保持 TCP 长连接，但并没有消息通讯（比如 GPS 等 IoT 设备与服务器之间的通讯），此时 1 千万并发连接下，如果能够维持 1 万 RPS，这也是只有 epoll 才能做到的，poll/select 是不可能做到的。

第 9 课

Geek_007：看评论区，很多同学都说是长连接，普通的 HTTP keep-alive 会不会有坑，三大运营商或者中间网络设备都会将超过一定时间的链接 drop 掉。如果没有 H2 这种 ping 保活的机制，有可能客户端长链接莫名其妙的就被 drop 掉，客户端只能依赖超时来感知异常，反倒是影响性能了。

作者：是的，不只网络设备，一些代理服务器为了减轻自己的负担，也会把长连接断掉，比如 Nginx 默认关闭 75 秒没有数据交互的 keep-alive 长连接。

第 10 课

安排：TTL 每一跳减少 1，这些怎么和 MSL 对应起来呢？每一跳减少的 1 相当于 1 秒？

作者：不是，这是一个预估值，所谓每一跳，是指每经过一个路由器网络设备，将 IP 头部中的 TTL 字段减少 1，并不等于 1 秒，通常推荐的 TTL 的初始值是 64。

第 11 课

Trident：带宽时延积如何衡量呢？网络时延不是固定的，是要多次取样计算平均网络时延，然后估算出这个时延积吗？

作者：是的，需要多次取样做估算，再乘以带宽。

第 12 课

妥协：为什么报文 5 之后的 ack 都是 ack6 呀？

作者：TCP 是有序的字符流，因此接收方收完报文 5 后，只能接收报文 6，但现在却接收到了报文 7、8、9、10，此时接收方该怎么办呢？

当然，它可以当做不知道，什么也不做，坐等报文 6 的到来。报文 6 什么时候会到呢？RTO 时间超时后，发送方会重发报文 6，因为发送方一直没收到 ACK7！但是，RTO 是很长的时间，接收方直接反复地传递 ACK6，这样发送方就能明白，报文 6 丢了，它可以提前重发报文 6。这叫做快速重传！

第 13 课

有铭：“寻找宕机服务时，只要看队列首部最老的心跳包，距现在是否超过 5 秒，如果超过 5 秒就认定宕机。”这里的逻辑无法理解。如果要用这种方式检测心跳，那么肯定要不停地把队列首部的心跳包移除，让新的心跳包从尾部加入，那么如果这个加入的过程卡一点，岂不是就会误判？

作者：这种设计下，还必须限制每次移除心跳包的数量（分多次执行），以防止加入过程长时间得不到执行。而且，在这种极限场景下，必须监控 CPU 的使用率，如果长期维持在高占用率（可能你的集群规模已经超大，要每秒处理数百万心跳包），那么应当通过扩容更多的 CPU 核、分布式系统等其它方案来解决，这已经不是单台机器能解决的了。

第 14 课

东郭：请问老师，我在 Nginx 配置中，不管 ssl_certificate 和 ssl_certificate_key 是否配置 ecc 证书，抓包查看服务器的 server hello 响应中的 Cipher Suite 字段都是 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256，这是正常的吗？

作者：这是正常的，TLS 握手阶段 Nginx 的 Cipher Suite，要通过 ssl_ciphers 指令来配合支持的 suites，并可通过 ssl_prefer_server_ciphers 指定优先选用的算法。证书只是包含了密钥、身份等信息，它们与密钥协商方式、对称加密算法并无关系。

第 15 课

Geek_007：FB 也搞了一套压缩算法 ZSTD，对比起来也比 gzip 性能强很多，不清楚这些压缩算法的原理是啥？怎么对比？另外普通的 JSON 和 PB 有不同适合的压缩算法吗？怎么比较呢？

作者：

1. 压缩算法的原理都是基于香农的信息论，将高频出现的信息用更少的比特编码。虽然原理是一致的，但实现上却有很大的差别，比如 Huffman 通过建立 Huffman 树来生成编码，而 LZ77 却是通过滑动窗口，这就造成了压缩比、压缩速度都很不相同。
2. 比较它们的优劣，主要看 3 个指标：

   压缩比，比如 Brotli 的压缩比好于 ZSTD；

   压缩与解压的速度，比如 ZSTD 比 gzip 速度快；

   浏览器等中间件的支持程度，现在几乎所有浏览器都支持 Brotli（即 br），但 ZSTD 少有支持。

3. 普通的 JSON 和 PB 没有最适合的算法，还是要针对具体场景，比较方法参见我刚刚说的那 3 个指标。

今天的答疑精选就到这里，期待大家能一如既往的在留言区进行交流，如果有更多问题，也欢迎一并提出。

上一篇　加餐1 | 特别福利：陶辉视频课精选

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。