云风的 BLOG

思绪来得快去得也快，偶尔会在这里停留

« 策划们离不开的 Excel | 返回首页 | skynet 社区广州聚会小记 »

乐观锁和悲观锁

最近晓靖给 skynet 提了一个 pr 。

提之前我们讨论了好久，据说是因为查另外一个问题时改写了 skynet 的消息调度部分发现在某些情况下可以提高 CPU 的使用率。

之前 skynet 的消息调度采用的是基于 cas 的无锁结构。但本质上，并发队列这种数据结构，无论是采用 spin-lock 还是 cas 无锁结构，为了保证时序，进队列或出队列的部分都必须是依次进行的，也就是说，多核心无助于提高队列的性能。

使用无锁结构，无非是对发生冲突保有乐观态度，觉得大多数情况下冲突不会发生，一旦发生就采取重来一次的策略。

而使用 spin lock ，则是对冲突采取悲观策略，认为冲突经常发生，所以在操作共享字段时，锁住资源独享操作。

最终，都必须等前一件事情做完，才能接着做下一件事。

无锁结构的程序逻辑往往显得复杂，那么它的好处是什么呢？

无锁结构在乐观情况下，可以让处理过程尽量并行，只在可能发生冲突的那一刻才用系统的原子指令锁住一个字长的内存写入，然后立即放开。加锁和解锁是原子的，这样就可以回避死锁的问题。同时也不会因为 spin lock 锁住的指令过多（如果线程数多于核心数，就有可能在锁住的过程中发生线程挂起），而导致其它线程等待时间过长。

在 skynet 的核心消息队列调度模块中，无锁结构能获得的好处其实非常有限。为了简化代码，我们甚至不需要单独去锁队列的两端。因为在 skynet 的消息队列分两级。在运行过程中，如果此级队列有消息时，它根本不会进入全局主队列（用锁保证次序的那个队列）。

由于不再使用无锁队列，数据结构也可以大大简化。使用一个简单的单向链表就可以管理这个队列。之前为了解决队列长度问题，已经把次级队列设计成一个侵入式链表，这次要做的只是把那个数组部分去掉就可以了。

修改过后，一个明显的好处是突发的大量服务启动（往往是瞬间大量连接涌入造成的 agent 启动需求）变快了。这解答了之前的一个疑问 。

---

12 月 12 日补充:

这篇不在于讨论用乐观锁还是悲观锁的性能好坏. 上文最后提到的问题的解释, 并不是因为提高了锁的性能, 而是因为:

之前认为在 skynet 的多个 worker 竞争全局队列时, 抢到操作权是乐观的. 因为处理消息本身的时间远远长过操作全局队列的时间. 当多个 worker 竞争时, 我们认为几乎是不会冲突的.

所以在 worker 竞争全局锁时做了这样的处理: 如果失败, 就认为队列为空, 而不是重新尝试. 这样做可行是因为 worker 本身是对等的, 每个 worker 的职责是完全相同的。

一旦发生竞争，说明 worker 没有什么复杂的事情做（当消息处理时间足够短时，竞争的概率会上升），所以让竞争失败的 worker 暂时放弃更有助于节约系统的资源。即：如果任务不多，退化成单核处理更好。

这样实现倒不是为了提高性能，而是代码更简单（不需要再重试）。

这次发现在某些特定情况下，这个策略是错误的。至于改成 spinlock ，是因为代码量要小的多。而在队列实现上，无论采取怎样的策略，对系统的性能影响及其有限，那么选择一个代码更简单的实现更好。

云风 提交于 December 8, 2014 11:21 AM | 固定链接

COMMENTS

这也许是所有用了linux socket的程序都会有的问题。

Posted by: 丁大头 | (14) December 15, 2014 11:21 AM

可是要全数据流无锁必须得从底层开始就得分流了。
所以我好奇的是，skynet会不会遇到处理器再多也提升不了性能的问题。

Posted by: 丁大头 | (13) December 15, 2014 11:13 AM

skynet 主要解决的不是和外部通讯问题。所以它对外的 socket 处理全部放在单线程里完成，然后把这些信息分发到不同的线程里去做。

Posted by: cloud | (12) December 15, 2014 10:41 AM

当然了如果服务器核不多，DPDK的优势显不出来。至少8个物理线程以上才看得出。12线程以上很明显，几十个物理线程的那必须用。
当然如果上层逻辑占用处理器很多，应该会消耗很多空闲等待时间，会弱化这个缺点。

Posted by: 丁大头 | (11) December 15, 2014 12:35 AM

多谢解释。
我目前的理解是，使用了队列，就必须用锁或者原子同步。核很多时(>12)，这就构成了瓶颈。这是软件的做法。
而分流就类似于硬件做法，在驱动层就把网口收上的数据发往pcie的映射空间，各个核只查内存，负责收属于自己的数据（轮询中断皆可，中断有点慢）。Intel已经实现并开源了这个功能，叫做DPDK。问题是还没有开源的网络协议栈调用这些库，如果skynet基于linux socket实现，那就没法直接用DPDK。
我先去看看代码。

Posted by: 丁大头 | (10) December 15, 2014 12:31 AM

@丁大头

skynet 是两级队列的。网络线程会把消息投递到次级消息队列中。

如果投递到了一个活跃的队列里（工作线程正在处理）那么它就相当于投递到了那个核心上。

只有投递到无消息的队列中，那么才会引起一次一级队列的 push.

应该说 skynet 是尽量避免核心间竞争的吧。或者说，在系统繁忙的时候减少竞争。

欢迎读一下代码。

Posted by: cloud | (9) December 14, 2014 11:58 PM

@丁大头

我以为需要先实现一个队列，才能“把需要同步的放到一个核/物理线程上”。难道我搞错了？

Posted by: 杨博 | (8) December 14, 2014 10:09 PM

看了乐观锁悲观锁，以及以前的无锁编程，一直有个问题没想明白。
为什么不能使用先分流，把特定的包送到某个核，然后把所有的锁和原子同步都去掉？
我是做通讯处理器的，在通讯处理器，包括x86上写包处理程序的时候，第一原则就是避免锁。查找和修改全局表需要锁吗？不用，每个核维护自己的表。包与包之间需要通讯和同步？没关系，把需要同步的放到一个核/物理线程上。
我看到linux的网络协议栈就有一个大限制，由于受到rcu锁的影响，当物理线程大于12（或者6核xeon）的时候，性能就再也无法提高，100个核都没用。而解决的方法只有使用dpdk分流，原理和我之前说的一样。这样一改，性能就和物理线程数成正比了。github上有个类似的项目叫做fastsocket。
不知道skynet的网络协议栈也受此限制吗？

Posted by: 丁大头 | (7) December 14, 2014 07:43 PM

@spin6lock

因为那次的情况主要是因为工作线程高于实际核心数造成的. 而实际是不应该这么配置的.

这次改回去主要是这样权衡的:

当初改成无锁结构,是因为竞争导致的 cpu 空转, 导致外部连接无法处理是因为早期的版本中, socket 不是放在核心中,而是一个额外服务. 也就是和其它服务一起被调度的.

而现在 socket 已经是核心层模块,它有独立线程, 不参于消息队列的调度.也就是 socket 消息有最高优先级的处理能力.

而曾经使用的无锁结构(乐观锁) 在碰到竞争的情况时,调用方实际上是采取退让的策略, 这使得某些时候 cpu 的利用率又不够.

当然,我合并这个 pr 最主要的考虑是, 这样写足够简单, 没有明显的 bug .

Posted by: Cloud | (6) December 14, 2014 01:26 PM

http://blog.codingnow.com/2012/10/bug_and_lockfree_queue.html 云风，你这篇文章提到，似乎早期的skynet就用过spinlock，如果工作线程多，但实际工作比较少的时候，就会导致CPU占用率飙升，因为工作线程都忙着抢一个二级消息队列来取消息了。然后这个会导致玩家无法登录，为什么现在换回去用悲观锁，也不会有同样的问题出现呢？

Posted by: spin6lock | (5) December 14, 2014 11:04 AM

@杨博

后面我补充了. 这是一个特定情况下的 bug 。

因为是乐观的，所以之前的策略是如果失败就不再试了。

因为 worker 干的事情是一样的， 所以你不试总有人去做。

如果要保证时序的操作很简单（不涉及 IO 等操作), 用不用 cas 其实在乐观情况下差别也不会很大。

因为 cas 解决的问题是，我先把事情干了，然后再提交。这样多个核就有可能同时做事，只需要提交的时候按流水线正确的提交就可以了。这样可以提性能。

当然，我觉得 cas 主要解决的不是性能问题，而是避免死锁的正确性问题。

锁本身的性能问题也不足以干扰到系统性能，因为系统大部分资源应当在做锁以外的事。如果锁本身占据了系统可观的资源，那么系统的结构设计就是有问题的。

另外，skynet 加载脚本代码是没有 IO 的. 这个很早被优化掉了。

Posted by: Cloud | (4) December 12, 2014 02:50 PM

整篇文章我觉得没有大问题，对悲观锁和乐观锁的分析很好。

但这句话不合逻辑：

> 修改过后，一个明显的好处是突发的大量服务启动（往往是瞬间大量连接涌入造成的 agent 启动需求）变快了。这解答了之前的一个疑问 。

Posted by: 杨博 | (3) December 12, 2014 01:08 PM

我觉得一次IO（哪怕操作系统有磁盘缓存），和一次CAS指令相比，消耗时间恐怕不在一个级别上。

Posted by: 杨博 | (2) December 12, 2014 12:52 PM

不好意思，没看明白。

前文（http://blog.codingnow.com/2013/12/skynet_agent_pool.html）说，性能瓶颈在字节码加载上，我理解瓶颈在IO上。

这里又说跟worker竞争全局队列有关。

似乎两种说法不太一致？

Posted by: 杨博 | (1) December 12, 2014 12:48 PM

POST A COMMENT

非这个主题相关的留言请到：留言本

名字：

Email 地址：

为了验证您是人类，请将六加一的结果（阿拉伯数字七）填写在下面：

URL:

记住我的信息？

留言：
（不欢迎在留言中粘贴程序代码）

提交