分析报告

一．各种锁的原理

1. spinlock:

spinlock，也称自旋锁,是属于忙等待类型的锁。spinlock 最多只能被一个可执行线程持有。如果一个可执行线程试图获得一个已经被持有的 spinlock，那么该线程就会一直进行忙等待，也就是空转，等待锁重新可用。如果锁未被使用，请求锁的执行线程便立刻得到它，继续执行。

优点：spinlock 的设计初衷，在短时间内进行轻量级加锁。spinlock 适用于竞争不激烈，线程数较少，并且临界区小的情况。

缺点：一个被争用的 spinlock 使得请求它的线程在等待锁重新可用时自旋，特别的浪费 CPU 时间，所以 spinlock 不应该被长时间的持有。还会产生死锁，即如果一个已经拥有某个 spinlock 的 CPU 想第二次获得这个 spinlock，则该 CPU 将死锁。

当 spinlock 使用者保持锁时间非常短的时候选择自旋而不是睡眠是非常必要的，这时 spinlock 的效率远高于 mutex lock。但是 spinlock 一旦被持有时间过长，性能就会变差。如果需要长时间锁定的话, 最好使用信号量。

2. mutex lock

mutex lock 属于 sleep-waiting 类型的锁. 我们实现的 mutex lock 是基于 futex 的。futex 是由用户空间的一个对齐的整型变量和附在其上的内核空间等待队列构成. 多进程或多线程绝大多数情况下对位于用户空间的 futex 的整型变量进行操作。mutex lock 最多只能被一个可执行线程持有。如果一个可执行线程试图获得一个已经被持有的 mutex lock，那么其他线程就会进入睡眠。

优点：如果需要长时间锁定的话, 最好使用 mutex lock。防止 CPU 空转，提

供更好的性能。

缺点：使用当者保持锁时间非常短的时候，反复的睡眠唤醒会造成大量的在睡眠唤醒之间切换，时间反而变长。

3. two-phase lock

因为 spin lock 和 mutex lock 分别适合时间短的锁定和时间长的锁定，所以我们可以写出一种两阶段锁同时适用于这两种情况。它的原理是先自旋一段时间，如果可以获得锁就不再睡眠，减少了在睡眠唤醒之间切换的时间，如果在自旋结束时还是没有获得锁，那么就进入睡眠等待被唤醒。

优点：这种锁综合了两种锁的优点，同时适合时间短的锁定和时间长的锁定，虽然在特殊的情况下可能不如 spin lock 和 mutex，但是从综合考虑就有比较好的效果。

缺点：自旋的时间比较难以把握，而且针对不同时间的线程，自选的时间还应该有所变化才能取得更好的效果。

4. spinlock with ticket

每个线程都带有一个 ticket，作为自己独特的标志，即指定下一个进入临界区的线程，这样线程就会按照开始的顺序执行完。

优点：这种锁不会有“饥饿”产生，按照先来先服务的原则保证了先进先出，确保了优先级。

缺点：效率不高，比 spinlock 更慢。

二．关于锁的优先级

1. spinlock:

spinlock 本身没有关于优先级的设定，完全是线程随意抢占，这样就容易引发“饥饿”的现象，即有的线程可能会一直抢不到锁。

我们的处理方法是写了一个带有 ticket 的锁，即指定下一个进入临界区的线程，这样线程就会按照开始的顺序结束。这种写法按照先来先服务的原则保证了先进先出，不会有“饥饿”产生，然而效率也不够高。

2 .mutex lock

mutex lock 用的是 futex 自己维护的队列，根据我们查到的资料，可以使用 FUTEX_WAIT_REQUEUE_PI 和 FUTEX_CMP_REQUEUE_PI 来进行带有优先级的睡眠和唤醒。

三．关于图表的分析

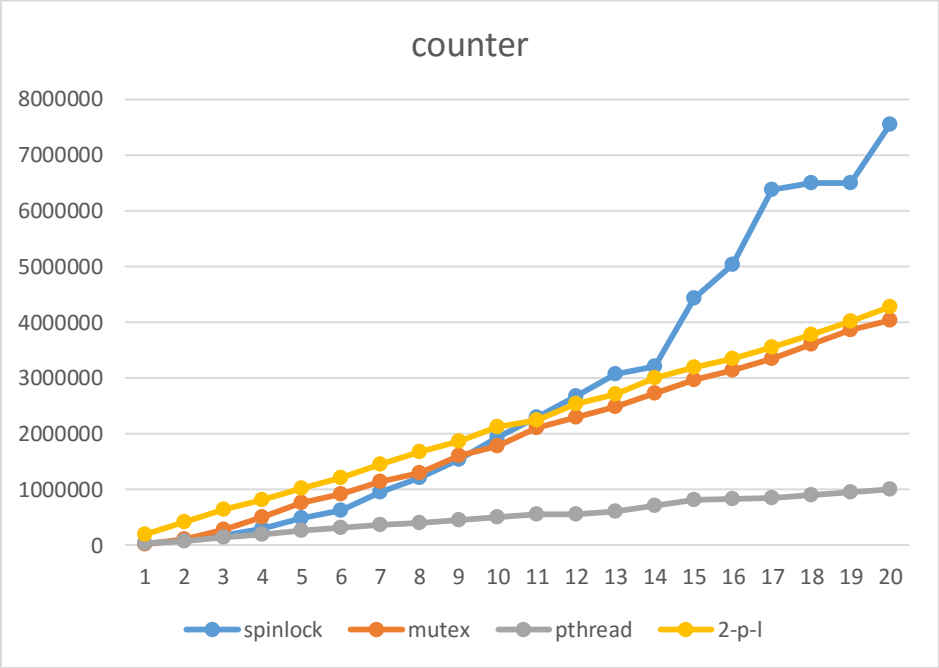
（LINUX 系统的 CPU 是 4 核）

三种数据结构统一的规律是：

随着线程数量的增多，时间都会增长，但几种锁时间的增长幅度不同，其中 pthread 一直速度最快，性能最好。而 spinlock with ticket 一直最慢，因为它除了空转还要保证线程按顺序执行。为了突出 spinlock 和 mutex lock 的区别，这里不再画出 spinlock with ticket，但它的性能是可以确定的。

其余的锁的性能和具体实现的数据结构以及线程的多少都有关系。从总体来看，two-phase lock 要略微优于 mutex lock，然而两者差距不大，spinlock 一旦遇到线程数量多或是时间长就会变得性能较差。

1. counter

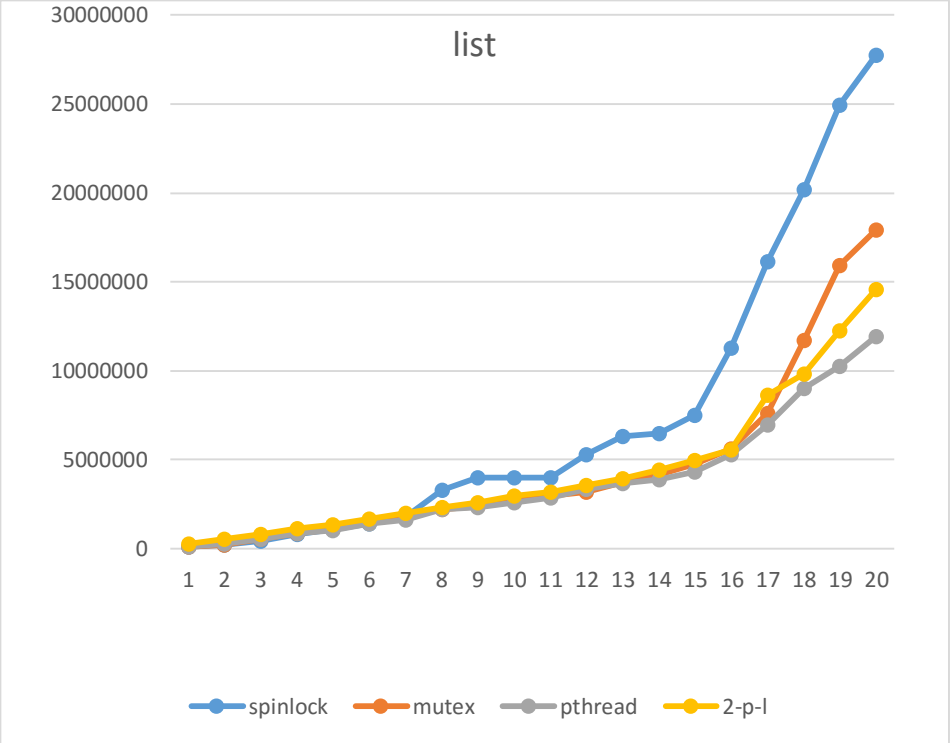


纵坐标：时间 （单位：微秒） 横坐标：线程数（单位：个）

测试内容：counter++ 循环 10^6 次

counter 这样的数据结构在调用锁的时候拥有锁的时间很短，在线程少的时候 spinlock 的效果要好于 mutex lock。但在线程数量多的时候 spinlock 的空转浪费了 CPU 时间，而 mutex lock 在睡眠和唤醒的时候是不占用 CPU 时间的，这时就体现出了优势。Two-phase lock 在这里和 mutex lock 的效果类似，开始比较慢，但是线程数量增多就取得了比较好的效果。但是 two-phase lock 不如 mutex lock，可能与空转相关。

2. list



纵坐标：时间 （单位：微秒） 横坐标：线程数（单位：个）

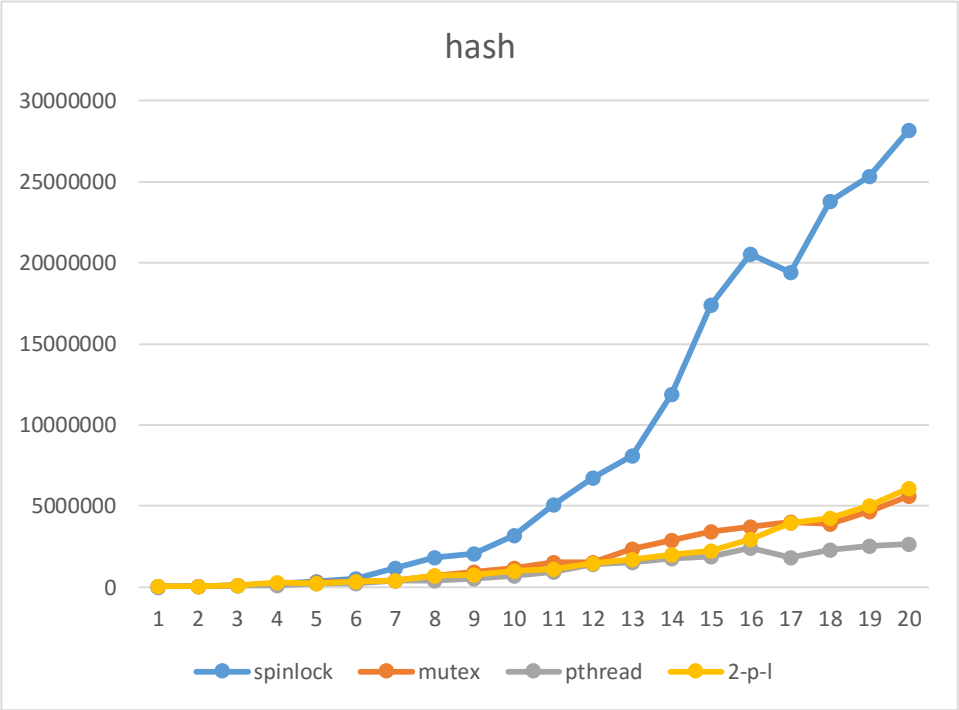
测试内容：list 插入节点 循环 10^6 次

list 的数据结构在调用锁的时候拥有锁的时间较长，所以 two-phase lock 和 mutex lock 的总体效果要好于 spinlock。在线程数量增多的时候，two-phase lock，mutex lock 和 spin lock 的时间都有明显的提升，原因是线程拥有锁的时间较长，竞争激烈，自旋和睡眠唤醒的时间都会明显增加。在这里 two-phase lock 体现出更明显的优势，更接近于 pthread。

这是我们只用 list-insert 函数测出的结果，由于是在头部加入新的节点，spinlock 和 mutex lock、two-phase lock 的差距不是很明显，和 counter 数据结构相比锁中的内容变长的不太多。所以在接下来的 hash 数据结构测试中，我们换了一种测试方法，除了加入，我们也测试删除和查找节点，这样会遍历整个链表，效果更加明显。

3. hash

3.1 线程数不同

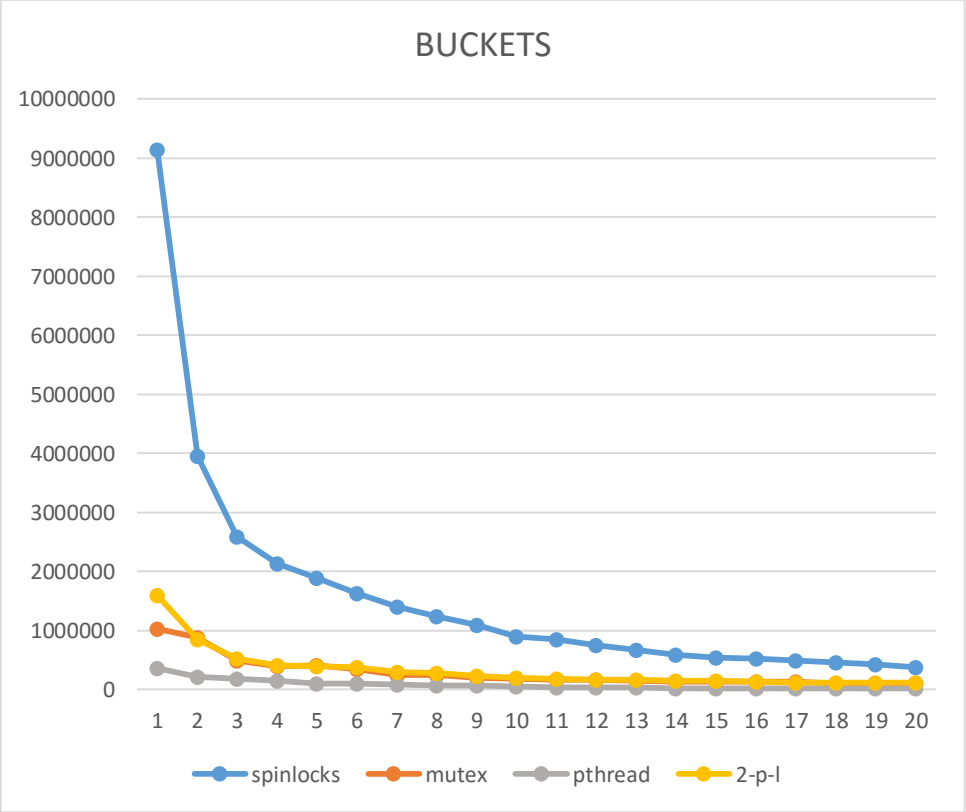


纵坐标：时间 （单位：微秒） 横坐标：线程数（单位：个）

测试内容：hash 插入、删除、查找节点 循环 10^4 次

由于 hash 的数据结构由 list 实现，在调用锁的时候拥有锁的时间较长，再加上我们也测试了删除和查找节点，这样会遍历整个链表，总体效果更加明显。mutex lock、two-phase lock 与 pthread 的差距不是很大，其中以集合了两种锁优势的 two-phase lock 性能更佳。而且由于持有锁的时间明显增长了，spinlock 在线程数还不是很多的时候在时间上已经有大幅度的增加，效率远远慢于其他几种锁。

3.2 BUCKETS 的值不同



纵坐标：时间 （单位：微秒） 横坐标：BUCKETS

测试内容：hash 插入、删除、查找节点 循环 10^3 次

线程数：20 个

BUCKETS 的值是用来给 hash 表分组的，BUCKETS 的值越大，hash 表分的组越多，在删除和查找节点的时候遍历的节点数就越少，删除和查找的速度就越快。所以时间都呈下降趋势。

和上一张图相同，mutex lock、two-phase lock 两种锁的性能非常接近，与 pthread 的差距不是很大。Spinlock 的性能不好，尤其是在 BUCKETS 数量很少，链表很长的时候，spinlock 要花费大量的时间。