Project 4　Locks and Threads

测试报告

10152130137 汪贻俊
10152130220 邢思远
10152130145 许　倩

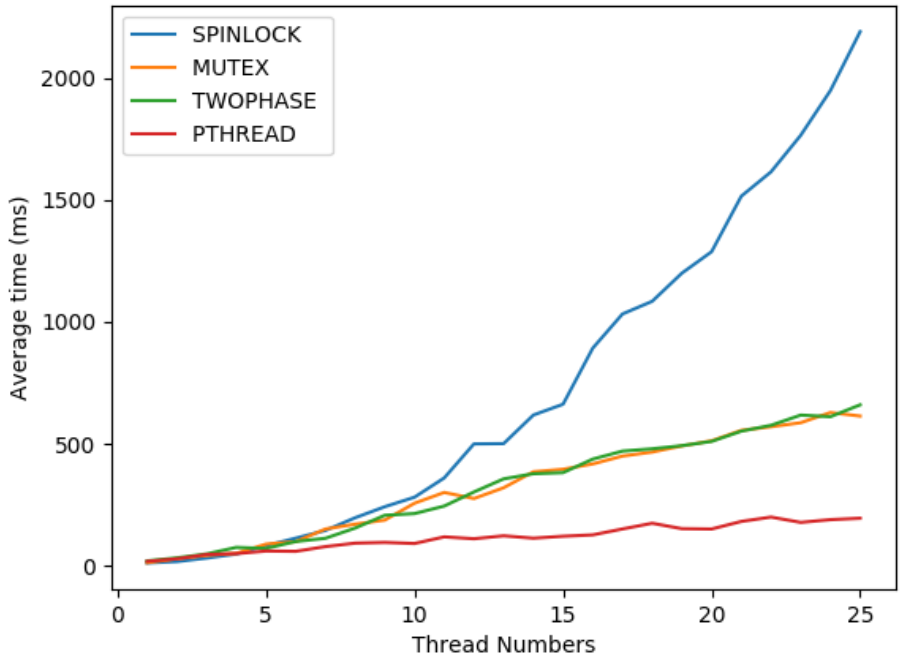
目录

1. Counter 测试

1.1 测试方式

Counter 在不同种类锁和线程数下, 每个进程进行 10^6 次加法操作所花时间。

1.2 测试结果



1.3 结果分析

每个线程进行 10^6 次的加法，随着线程数的增加，所消耗的总时间增加是必然的，因为加法的次数也在增加。但是可以明显地看到不同锁的条件下消耗时间增长的速度有显著的差异，SPINLOCK ＞ MUTEX~TWOPHASE ＞ PTHREAD, 因为 SPINLOCK 是通过不断循环来等待获得锁，因此在进程数量增加时，导致 CPU 大量空转，浪费了大量时间。而 MUTEX 和 TWOPHASE 的结果很相似，因为两者最主要的都是使得未获得锁的线程睡眠，从而减少时间浪费，当然系统的 PTHREAD 的表现是最好的。

2. LIST 测试

2.1 测试方式

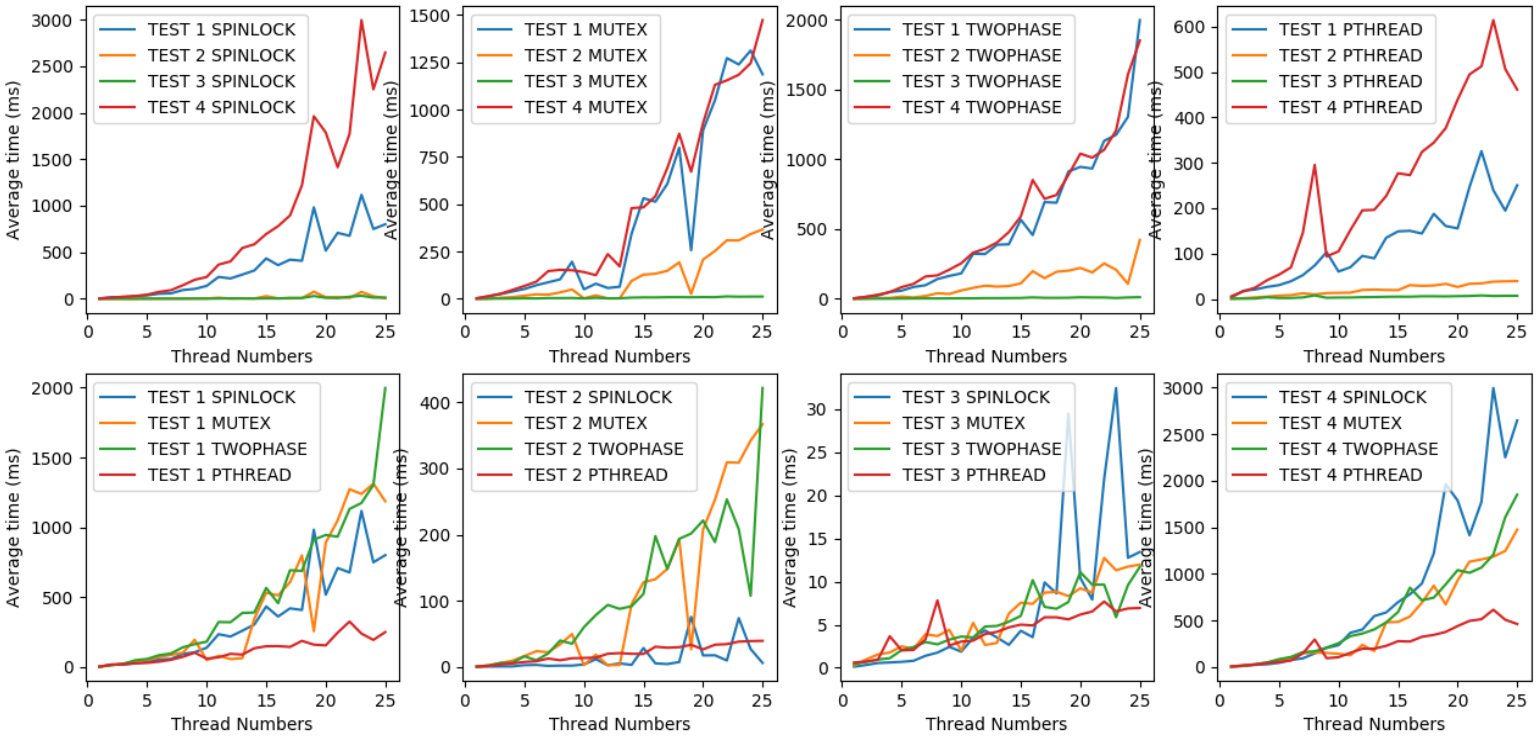
TEST1：按顺序插入 1000 次，同样顺序删除 1000 次

TEST2：按顺序插入 1000 次，倒序删除 1000 次

TEST2：重复插入删除 1000 次

TEST4：按顺序插入 1000 次，随机删除 1000 次

2.2 测试结果



2.3 结果分析

1） 对于 LIST 的测试我们首先测试每种锁对于 LIST 不同操作情况下的表现，从得到的测试结果可以看出：对每种锁，随机删除测试对于锁的都是最大的，然后就是顺序插入顺序删除操作，之后就顺序插入倒序删除，最后对于插入

后立即删除的情况每种的锁的消耗都很少，这几种操作的差别主要在查询时间上，不同的删除方式导致线程执行删除时所花费的时间差异很大。

2) 在每种情况下不同锁的对比在第二行的四幅图可以看出，SPINLOCK 在 TEST2 中的表现相当出色，而在 TEST4 随机删除的测试中表现却很不好，同样反映了对于操作复杂时间较长，上下文切换较多的情况下，由于一直处于用户态，SPINLOCK 的表现会优于其他锁。而 PTHREAD 在四个测试中的表现都是很好的。剩下的 MUTEX 和 TWOPHASE 这两种的锁的表现相近，TWOPHASE 要略微优于 MUTEX，可能还是 SPIN_TIMES 的设置还需要探究，以期达到更好的效果。

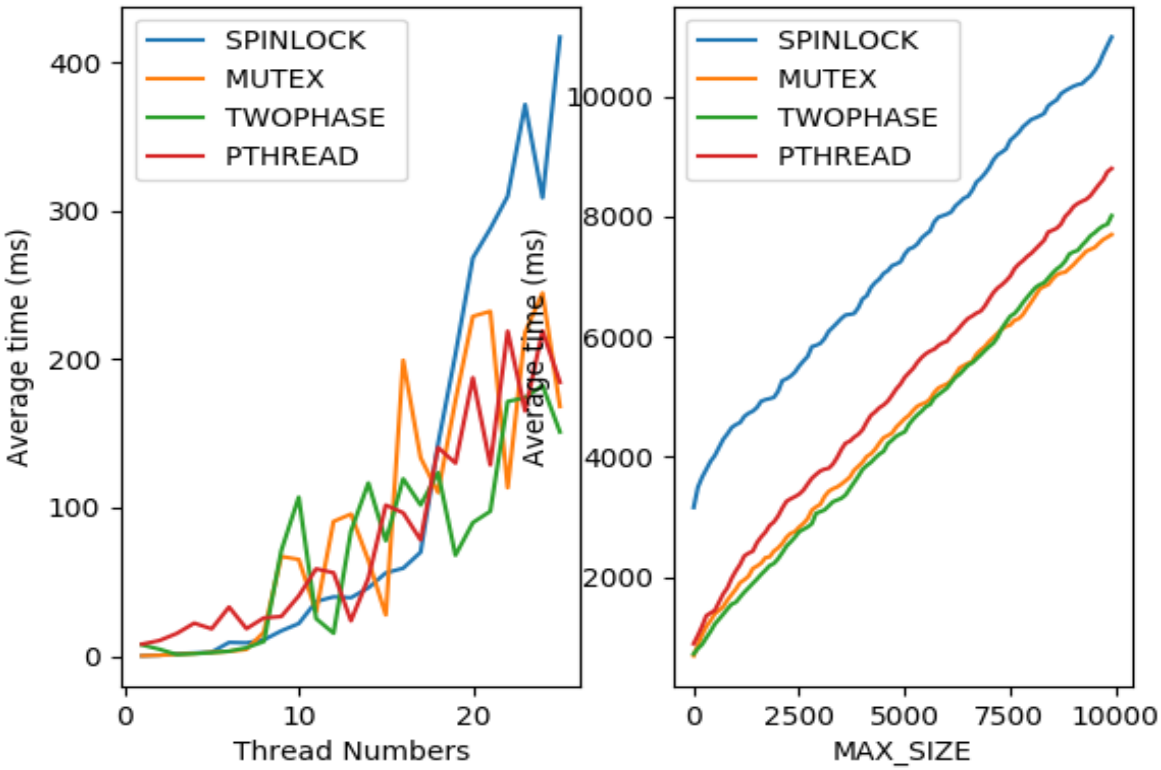
3. HASH 测试

3.1 测试方式

1） 不同线程数下，不同种类锁，SIZE 固定为 100，插入删除 10^6 次的时间。

2） 线程固定为 20，不同 SIZE，不同种类锁，插入删除 10^6 次的时间变化。

3.2 测试结果



3.3 结果分析

1） 对于链表的插入删除，在插入删除 10^6 次的测试中，随着线程数量的增加，时间总体趋于增加是肯定的，但是这个测试中不同锁的表现相对比较比较复杂。从整个的变化趋势来看，SPINLOCK 的表现在线程数较少时是有优势，因为线程数相对较少时，其他三种采用 sleep 的锁会有较多的上下文切换的时间花费，但是随着线程数的增加，这种优势就不存在了，SPINLOCK 后期

花费的时间依然是最高。至于其他的三个锁，表现基本相近，因为涉及到链表的插入删除还有查询，所需要的操作明显增多，MUTEX、TWOPHASE、PTHREAD 的核心都是对未获得锁的线程让其 sleep, 只不过 sleep 之前所留的时间是不同的，在这个测试中，可能是因为系统环境的不确定，这三种锁的表现也是不确定的，最后表现基本差不多。

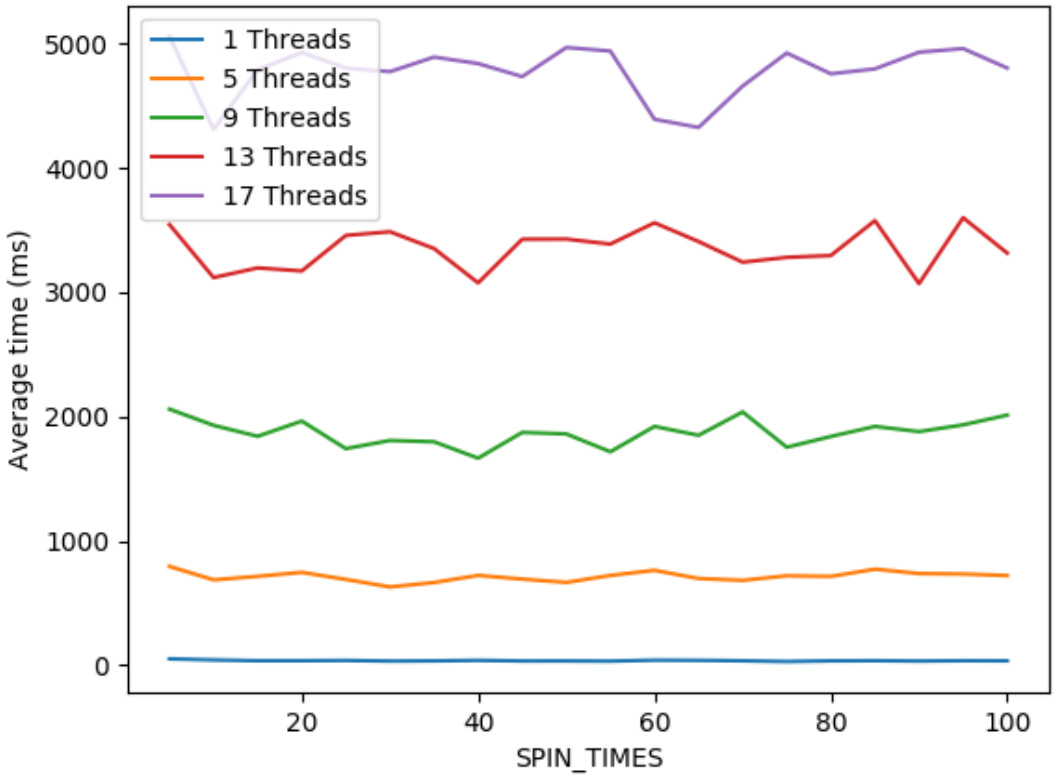
2) 在线程数量固定在 20 时，每种锁随着 HASH 表的 SIZE 变大，完成一次测试的时间都在变长。理论上来说，SIZE 变大，由于冲突的减少，速度应该是变快的，但实际测下来确实速度变慢。因此我们推测可能是由于 SIZE 变大，节点增大，需要维护更多链表，每个链表由于要并行，花在锁上的增多时间大于由于冲突减少而节省的时间，所以随着 SIZE 变大，总体效率变慢。

4. TWOPHASE 测试

4.1 测试方式

TWOPHASE 锁在不同线程数，不同 SPIN_TIMES 下完成 10^6 次 Counter 插入操作的效率。

4.2 测试结果



4.3 结果分析

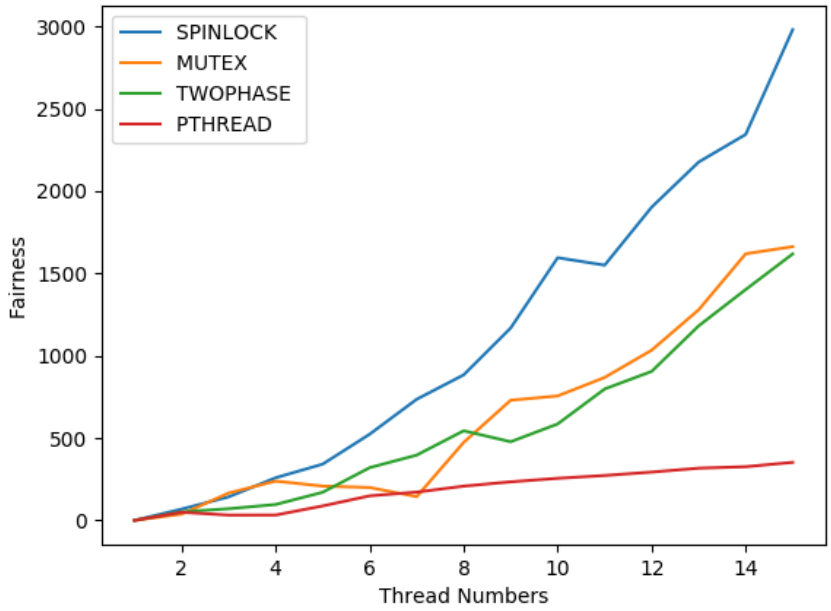
从测试图可以看出：因为 TWO_PHASE 锁分为两个阶段，第一个阶段需要 SPIN，那 SPIN 的次数就显得尤为关键，不过这次的测试效果不太理想，因为系统环境的变化，不同情况在 SPIN 的次数也是很难确定最好的效果，最终测试结果也无法确定一个最好的 SPIN_TIMES。

5. 公平性(Fairness)测试

5.1 测试方式

将公平性量化为，对于某种锁，某个线程数下，对于各个线程完成 Counter 插入 10^6 次所花时间的标准差。这个数值越小公平性越好。

5.2 测试结果



5.3 结果分析

在进程数增多时，不同的锁在公平性方面的差异还是挺明显的，SPINLOCK 在公平性上的表现是最差的，因为 SPINLOCK 没有做任何不同线程调度顺序上的管控，只要锁被释放，所有的等待的线程都会来竞争锁，没有对锁的调度的管理，因此公平性是最差的。而 MUTEX 和 TWOPHASE 都是使用 Linux 提供的 futex 系统调用来实现的，会在内核态维护一个等待线程的队列，从而达到管理线程的目的，公平性的表现就好了很多。系统的 PTHREAD 在公平性上的表现也是最优的，系统通过线程的调度管理，很好的避免了“饥饿现象”。

6. Conclusion

从以上的所有测试可以基本总结出这四种锁的比较结果：

与系统的 PTHREAD_LOCK 相比，我们实现的三种锁还有一定的差距，对于 SPIN_LOCK, 如果执行的操作比较复杂，消耗时间较长，相比于产生大量上下文切换的其他锁相比是有一些优势的，但是遇到线程较多的情况，SPINLOCK 的线程调度是混乱的，无法保证公平性，无法有效地管理线程，所以与其他三种锁相比还是差距较大的。

至于 MUTEX 和 TWOPHASE，这两种锁的核心都是使用 sleep 来减少空转，它们的公平性的表现得到较大的提升，但是 TWOPHASAE 的关键 SPIN 次数的设置还需要继续探索，可以设计一套动态改变 SPIN 次数的算法来提高 TWOPHASE 的性能，这样 TWOPHASE 的表现会更加接近系统的 PTHREAD_LOCK。