Report On Linux-Lock

Non-scalable locks are dangerous

本文主要阐述了不可扩展自旋锁（Non-scalable spin lock）的性能问题，发现了自旋锁随着核数目的提升会产生崩溃（collapse）现象，分析了这种现象的原因，建立了用于分析自旋锁性能的Markov模型，并通过实验分析了不可扩展自旋锁和可扩展自旋锁（scalable spin lock）之间的性能差异。

**本文主要创新点如下：**

（1）发现了排队自旋锁在核数增多的情况下会出现崩溃现象；

（2）建立了基于Markov-Chain的自旋锁分析模型；

（3）实验验证了可扩展自旋锁MCS在linux下的性能优于不可扩展自旋锁，但是可扩展自旋锁之间性能差距不大。

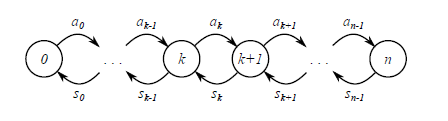
**本文的解决思路如下：**

首先，作者通过四组benchmark设计实验验证了不可扩展自旋锁的“在有高竞争的情况下，性能大幅度下降”的性质。实验发现，当在关键区段（critical section）比较短时，自旋锁的性能会在核数目升高到某一个阈值的时候快速下坠。这是本文面向的问题。

（Q： 文中出现了critical section和serial section这两个名词，通过前后文来看，这两个概念应该是一致的，但是我不太清楚其具体指的是那一段代码？）

文章对这个现象进行了定性的分析，认为自旋锁的工作流程中，一旦发生竞争，系统需要进行大量的核间操作来选择下一个获取锁的核，而这个过程是导致性能下降的主要原因。（理解：实际上这是一个正反馈的过程，一旦由于多核竞争导致性能下降，就会在这段期间内累积更多的等待核，从而竞争更加剧烈，性能更差，最终导致性能崩溃。）

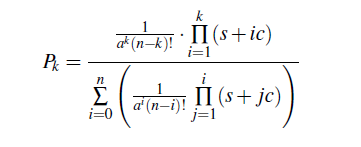
为了定量分析，该工作进行了建模。首先，文章阐述了系统使用的cache一致性协议（cache coherence protocol），选择了最简单的MIS三状态协议，即，Modified、Invalid、Shared三个状态。此外，作者还描述了多核通过主目录（home dictionary）来申请访问cache的机制，列出了在不同的cache状态下，主目录对于多核访问的响应类型。在这个模型的基础上，文章建立了基于Markov的模型，如下图所示。



其中，每一个状态描述了处于等待状态的内核的数量，其稳定状态概率是，是新的核进入等待的概率，是自旋锁释放并由下一个核获取锁的概率。（这两个概率由相应的时间来确定，因此应该是基于泊松过程的）这两个概率可以表示如下公式：

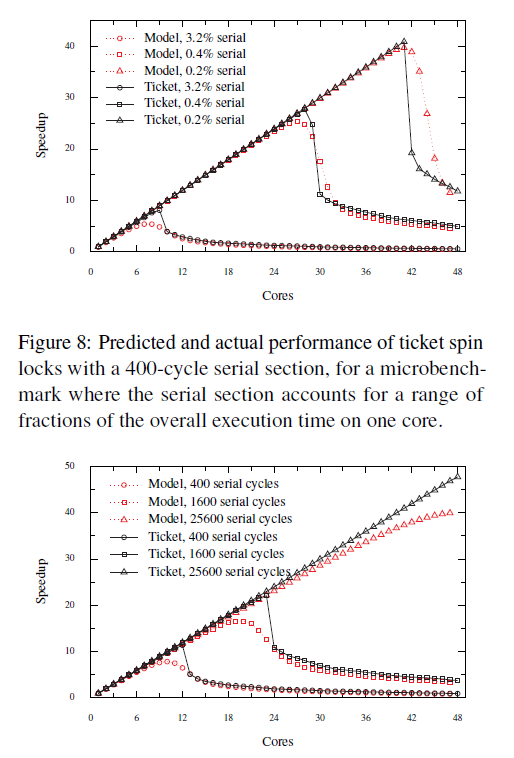
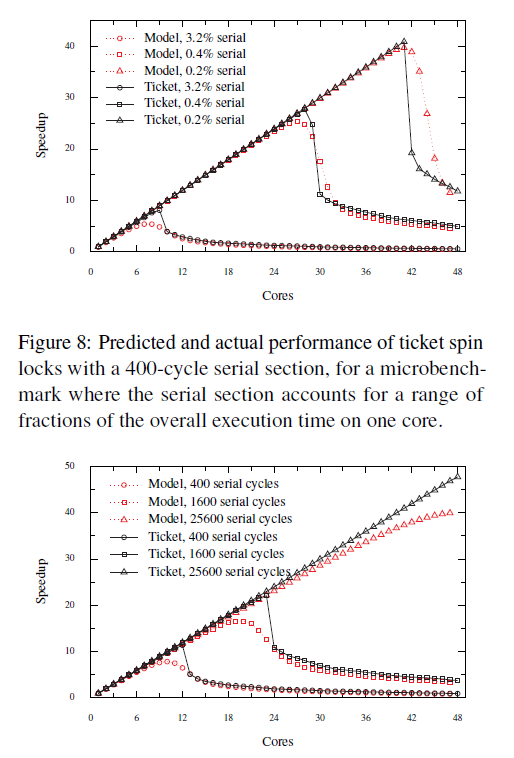
当状态稳定时，状态转入和状态转出的概率一致：

因此可以推出公式的表达式：



其中c是关键区段（serial section）的长度。因此可以看出，关键区段的长度越短，越有可能产生崩溃。

文章的最后完成了两个实验。其一是用于证明模型的准确性：



两组实验分别是固定了关键区段的出现次数和其长度。

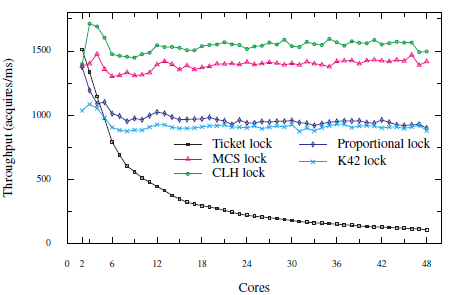
另一个实验验证了可扩展自旋锁的性能。可扩展自旋锁的特点是每次申请时cache缺失次数是固定的，其实现有两个要点：

（1）存在一个队列来管理等待状态的核（waiter）

（2）每一个核在自己的队列入口处自旋（spin）

（没太明白第二条的意思，通过文中对于MCS的描述，是说：waiter构成了一个链表，每一个链表节点是一个pnode，每一个内核等待时都在pnode上自旋，即执行轮询操作。这和传统的spin lock有什么区别？是说传统的SL是逐次轮询的吗？这为什么会影响cache的命中次数呢？）

作者还对传统的自旋锁进行了补偿，并选了最优的补偿系数来与其他的可扩展锁进行对比。



实验的结果是MCS的性能优势要高于不可扩展自旋锁3.5~16.0倍。

**本文的主要相关工作是：**

（1）T. E. Anderson. ‘90关于自旋锁性能下降。

（2）S. Boyd-Wickizer et. al. ‘10发现了崩溃现象，提供了简单的模型。

（3）J. M. Mellor-Crummey and M. L. Scott ‘91给出了分析性能的关键公式。

**本文结果的验证指标：**

即性能的验证指标，包括正常工作的核的数目和吞吐率。

**本文的不足之处：**

从文字上来看，感觉没有讲清楚一下两点：

（1）可扩展自旋锁的优势在什么地方？为什么它的结构会导致这样的优势？

（2）对于排队自旋锁的可扩展性修正，是根据等待核的数量进行补偿。那么补偿的量是什么，选的参数有什么含义？