Non-scalable locks are dangerous

阅读报告

江梦　P14206009

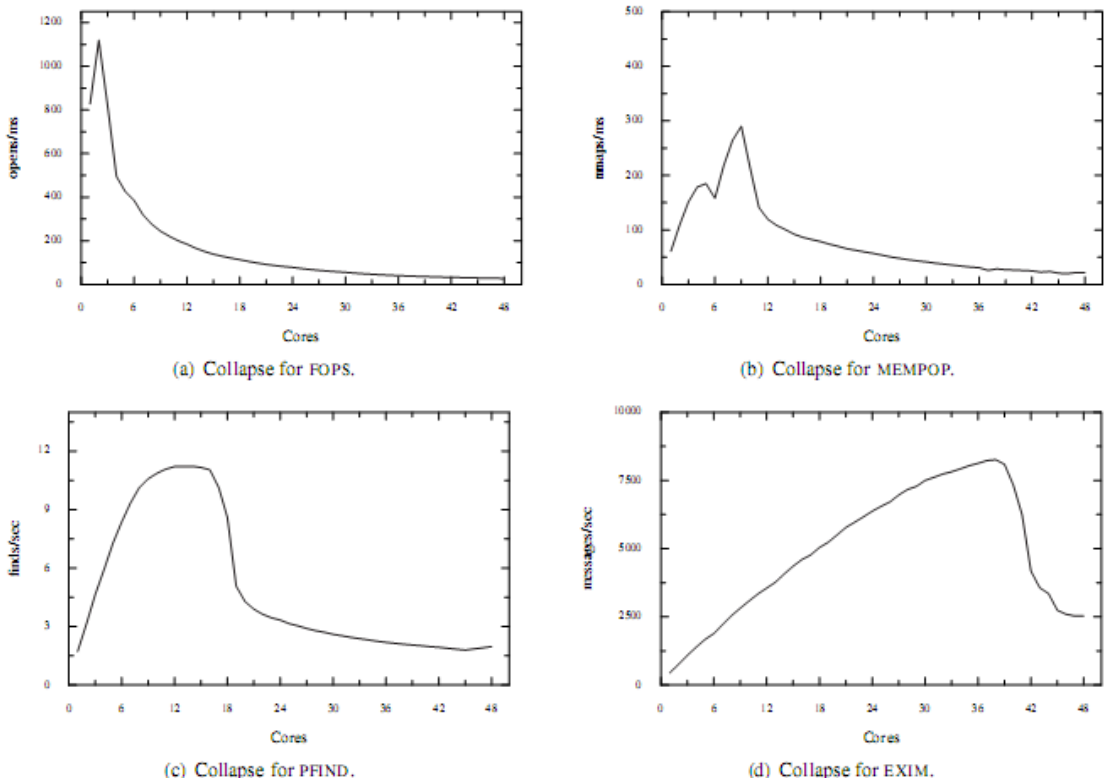
有些操作系统的串行化依赖非扩展性的自旋锁，如 Linux 内核使用了 ticket 自旋锁。本文说明了这些非扩展性的锁在实际工作中会引发巨大的性能崩溃，并提出了一种新的基于 Markov 的性能模型来解释这种崩溃的本质和突发性。此外作者还阐述了用可扩展的自旋锁来替代这些非扩展性的自旋锁能来避免崩溃的方法。

众所周知，非扩展性锁在高度争用时性能很低，但还是有很多系统采用非扩展性锁。由此带来的突发性的系统崩溃也很常见，例如 25 核时运行良好的系统在 30 核时就完全崩溃，同样令人吃惊的是这种情况下攻击性临界区通常也很小。本文通过对 Linux 内核中锁的研究分析论述了非扩展性锁是危险的，主要分为三个方面：首先，非扩展性锁会严重降低整体性能并很可能发生于实际系统中；其次，随着核数增加，性能崩溃可能突然开始，例如系统 N 核时有良好性能但增加几个核后总性能大大降低；最后，临界区表面上看来似乎非常短，也许仅有几条指令，但也会触发性能崩溃。本文提出了一种对非扩展性锁性能的预测模型，来解释上述几种现象。由此，作者认为系统应该使用可扩展性锁，尤其是像操作系统内核这种工作负载和争用级别都很难控制的部分。文中示范了用可扩展的 MCS 锁取代 Linux 中的自旋锁的实例，并通过重演引起性能崩溃的软件来测试可扩展性锁的效果。此种方法也有些异议，如通过修改软件来消除串行瓶颈的非扩展性行为是确定不变的，而可扩展性锁只是推迟了这种修改需求。这个理论是正确的，但实践中不可能一次消除内核中所有潜在的争用点，即使内核在某个时间点上可扩展性很强，但其后的硬件很可能导致瓶颈。因此，我们可以将可扩展性锁看作一种通过更基础地提升内核扩展性来缓解时间危急程度的方法。

本文首先阐述了实际工作中的非扩展性锁会引起性能崩溃。作者以 Linux 内核使用的 ticket 锁为例说明了非扩展性锁的工作原理以及会引发的问题。2.6.25 版本之后的 Linux 内核都默认使用 ticket 锁，它有两个字段：当前占用锁的票号(current_ticket)和下一个未使用的票号(next_ticket)，需要获取锁的核使用原子指令对 next_ticket 增一来获得票号，然后该核保持自旋转直到其票号与 current_ticket 相等则成功获取锁，释放时核对 current_ticket 增一使锁交给下一个等待的核。等待锁的核都有锁变量缓存，一旦解锁这些核都将读取缓存，多数架构中这些读取是串行化的，因此读取操作所需时间与核数呈正比。锁等待队列中的下一

个核会在这个过程中获得其锁变量缓存的副本，因此每次切换锁的花费也与核数呈正比，则自旋锁每次释放的成本可以设为 O(N)。

作者在一个采用 Linux 内核 2.6.39 版本的 48 核机器上从 4 个基准方面测试了自旋锁，分别是 FOPS、MEMPOP、PFIND、EXIM。一种常见的想法是随着核数量增加总吞吐量呈正比增长，然后由于连续切片趋于平稳，但是测试结果（如图 1）表明一定时间内吞吐量随着核增多而上升，之后并不会趋于平稳而是在增多几个核后急速下降，往往在 N 核时拥有良好性能而增加 1 或 2 个核后性能会急剧下降。



(a) Collapse for FOPS. (b) Collapse for MEMPOP.

(c) Collapse for PFIND. (d) Collapse for EXIM.

图 1：Ticket 锁中的突发性能崩溃

图 1 所示的 4 个基准测试结果有相同特征，由此作者提出了以下问题：

（1） 为什么崩溃这么早发生？多数认为崩溃会开始于一个重要时机，如当许多核在同一时间需要获得该锁时，那么 MEMPOP 应该在 14 核左右性能降低，然而崩溃却更早发生——9 核时开始崩溃。

（2） 为什么最终性能降低这么多？

（3） 为什么性能崩溃如此快？由于每新增一个核会使瓶颈锁的释放时间少量增加，因此预期想法是随着核增多性能会缓慢下降。但结果并不如此，只是增加几个核也可能引起性能暴跌，即 N 核测试性能良好的系统可能在 N+2 核时运行很差。

为了回答这几个问题，文章引入了一种基于马尔科夫的模型，以便解释非扩展性锁的性

能崩溃行为。作者首先概述了硬件缓存一致性协议，并在此协议的基本特性的基础上构建了一个用于理解排队自旋锁的性能的模型，该模型能近乎准确的预测之前测试中的崩溃。构建模型需要考虑两种情况，一是没有争用时可以快速获得锁，二是在很多核争用锁时拥有权的转换所花费的时间会显著增加，此外，锁行为变化的确切点取决于锁的使用模式和临界区的长度等参数。为了能精确表示 ticket 锁的行为，作者提出了一个基于马尔科夫链的排队理论。如图 2 所示，链中的不同状态表示等待锁的核数，由于系统中假定有 n 个核，共有 n+1 种状态。每对状态之间到达率和服务率，分别表示获得锁的新核到达和锁被释放的行为，这些互不相同的速率能表示非扩展性 ticket 锁的性能并反映出系统是闭合的。从 k 到 k+1 的到达速率 ak 应该与系统中剩余的不等待锁的核数呈正比，相反，服务速率 sk 与 k 成反比，由于锁的转换时间随着等待核数线性增长。用 a 表示单核下获得锁的平均时间，则没有争用时单核的到达速率为 1/a，有 k 个核等待锁是 ak=(n-k)/a。计算服务速率时定义了两个参数：s——连续切片花费的时间；c——主目录对缓存行请求做出出回应的时间，根据缓存一致协议，缓存行的主目录要依次回应每一个请求，若有 k 个核请求读取锁的缓存行，则胜出者获得该缓存行的平均时间是 ck/2。处理连续切片和将锁移交给下一个等待核共花费时间 s+ck/2，因此服务速率为 sk=1/(s+ck/2)。虽然这个模型能准确的表示 ticket 锁的行为，但无法提供行为的简单公式。为此，用 P0,…,Pn 分别表示争用锁的核数即状态 0 到 n 的稳态概率，意味着转换速率平衡：Pk9 * ak = Pk+1 *sk，由此得出$\mathrm{Pk} = \mathrm{P0} \times \frac{n!}{a^k(n-k)!} \times \prod_{i=1}^{k}(s+ic)$。由于$\sum_{i=0}^{n} Pi = 1$，则$\mathrm{P0} = 1/(\sum_{i=0}^{n}(\frac{n!}{a^i(n-i)!}\prod_{j=1}^{i}(s+jc)))$，所以$\mathrm{Pk} = \frac{\frac{1}{a^k(n-k)!}\times\prod_{i=1}^{k}(s+ic)}{\sum_{i=0}^{n}(\frac{1}{a^i(\mathrm{n-i})!}\prod_{j=1}^{i}(s+jc))}$。从而可以计算等待核的平均数$\mathrm{w} = \sum_{i=0}^{n} i \cdot \mathrm{Pi}$，则到达速度为 n-w。

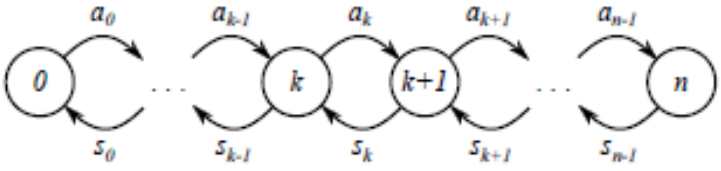


图 2：n 核的 ticket 自旋锁的马尔科夫模型

作者通过比较单核上微基准测试得到的实际速度与模型的预测速度，来验证所提出的模型，分别测试了两种情况，一是固定连续切片的执行周期而总体执行周期递增，二是临界区占总执行时间的比不变但连续切片花费时间递增。测试结果中模型基本与实际速度匹配，表明该模型能准确地捕捉导致 ticket 锁性能崩溃的相关因素。但预测速度与测量速度存在一定的差异，即相比实际观测到的崩溃预测结果中崩溃略微平缓，原因是崩溃点附近 ticket 锁的

性能很不稳定，而模型预测的是平均稳态行为。此模型预测的行为提供了三个重要启发：

（1） ticket 锁出现的快速崩溃是其设计本身内在性质引起的，并不是由于实验硬件存在性能问题，任何与此模型匹配的缓存一致系统都会出现类似的大幅性能下降。图 2 所示的模型可以解释这种现象，如果一个锁累积了大量的等待核，随着 k 增加服务速率 sk 降低迅速，那么减少等待核数需要花费很长时间。因此，一旦锁进入竞争状态，更多等待核到达比缩减当前等待队列更容易发生。简单的说，移交锁的时间随着竞争的核数增多呈线性增长，而这个时间会增加临界区周期长度，因此越多的核争用这个锁，临界区周期越长，就越可能有新核加入对锁的争用。

（2） ticket 锁只有在连续切片短的时候才会发生性能崩溃。sk 在长度不同的连续切片中降低速度不同，如连续切片时间 s 很小时，sk=1/(s+ck/2)主要受 k 影响。即锁的获取和释放操作越少，其性能对应用总吞吐量的帮助越小。

（3） ticket 锁的性能崩溃使应用程序无法达到 Amdahl's 规律预测的最大性能。

以上说明了非扩展性锁会引发性能崩溃及其原因，因此作者分析评价了几种基于 x86 多核处理器中的可扩展性锁，以便选择一种来替代非扩展性锁。可扩展的锁为每个获取生成一个固定量的缓存丢失从而避免非扩展性锁中存在的崩溃。这些可扩展性锁都维护一个等待队列，其中每个等待的核都自旋于自身的队列条目，不同之处在于每个锁维护队列方式不同以及锁定和解锁 API 的不同。作者介绍了几种可扩展性锁，如 MCS lock、K42 lock、CLH lock、HCLH lock，并测试了它们的应用性能，如图 3、4 所示。

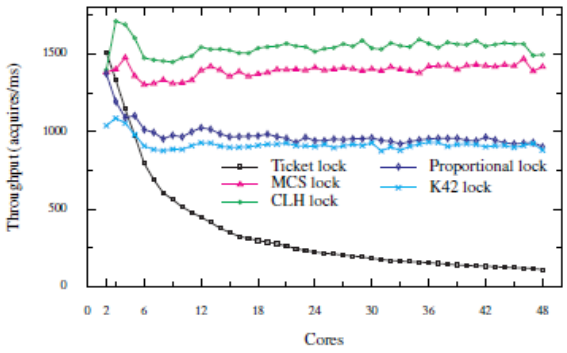


图 3：不同锁在多核争用时的吞吐量

Lock type	Single acquire	Single release	Shared acquire
MCS lock	25.6	27.4	53
CLH lock	28.8	3.9	517
Ticket lock	21.1	2.4	30
Proportional lock	22.0	2.8	30.2
K42 lock	47.0	23.8	74.9

图 4：不同锁的性能比较

基于上述测试结果，作者选择 MCS 锁来替代非扩展性的 ticket 锁，并修改了 Linux 内核中约 1000 行代码来布置 MCS 锁。由于 MCS 锁与 Linux 内核的 ticket 锁有不同 API，例如获得 MCS 锁的核要传一个 qnode 变量给 mcs_lock，释放锁时将相同变量传给 mcs_unlock。为此作者在栈中配置了 MCS qnode 变量，用 mcs_lock 和 mcs_unlock 代替原本 ticket 锁中的 spin_lock 和 spin_unlock，并将 qnode 传给 MCS 的获得和释放函数。在 Linux 内核中布置 MCS

锁后，作者从四个基准方面进行了测试，结果如图 5 所示，用 MCS 锁替代 ticket 锁使性能提高了 3.5 倍甚至特定情况下 16 倍之多。这说明使用可扩展性锁不需要太多代价，且能避免非扩展性锁带来的性能崩溃。

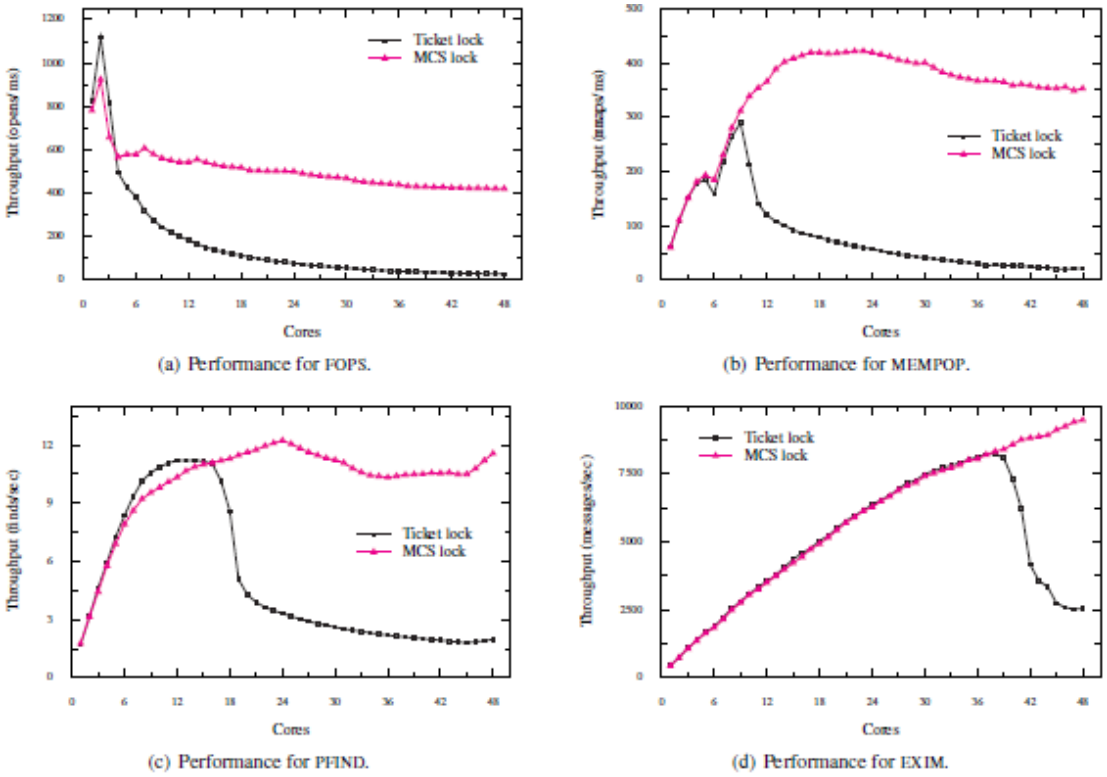


图 5：MCS 锁和 ticket 锁在 4 个基准测试中的性能对比

本文研究主要是可扩展性锁和非扩展性锁，这个领域已有大量文献，很多从业人员也熟悉这个问题，并且非扩展性锁在争用时表现不佳也是众所周知的。对于非扩展性锁，许多研究人员和从业人员都编写了微基准测试来表明现在硬件上非扩展性锁在争用时性能很差。而本文证明了非扩展性锁在看似合理的工作负载下会引起系统性能崩溃，也就是占有很短临界区的锁可能其内核执行时间规模很大。之前 Linux 扩展性研究已经说明了 EXIM 基准测试中的 ticket 锁会引起性能崩溃，但没有解释原因，本文展示了几个基准测试中崩溃的突发性并提出了可以解释崩溃的模型。此前 Eyerman 和 Eeckhout 提出了解释临界区相关的平行应用的闭合公式，并说明了一种机制使应用程序能达到优于 Amdahl 定理预测的速度，但他们的模型区别对待争用和非争用机制并分别提出了公式，这些公式并不能适用于锁的内部。本文提供了一个全面的模型，可以准确预测从非争用到争用整个过程的性能，并适用于锁内部。

此前的研究已经很好地说明了非扩展性自旋锁的低性能，但很少注意到它们的不良性能会对系统整体性能产生巨大影响。本文表明，非扩展性锁在实际工作中会引起系统性能崩溃，这种崩溃往往是突发的，并且本文还提出了一个 Markov 模型来解释这种突发崩溃。总之，

使用非扩展性锁是危险的，因为一个在 N 核测试时运行良好的系统很可能增加几个核后就发生性能崩溃，换句话说，如果升级一个机器以获得更好性能，面临着以性能崩溃结尾的风险。避免性能崩溃的一种方法是使用可扩展的锁，但在增加核来实现更高性能时需要设计新的数据结构。

本文的主要贡献是扩展了之前关于非扩展性锁有危险的研究，说明了非扩展性锁不仅仅是性能不好，还会引起整个系统性能的崩溃。具体而言，本文主要有三方面贡献：

（1） 文章阐述了实际工作中，非扩展性锁的不良表现会导致整个系统的性能崩溃，即使自旋锁只占用了内核中很短的临界区；

（2） 提出了一个简单又全面的模型来表现非扩展性自旋锁的行为，可以捕捉锁运行中的所有状态和操作。

（3） 文章验证了在基于 x86 的现代多核处理器上，MCS 锁可以在不牺牲性能的同时提高扩展性，并得出结论——不同类型的可扩展性锁之间扩展性和性能优势的差别很小。

我在读这篇论文的时候，认为作者将非扩展性锁的工作原理和低性能，以及它会引起系统性能崩溃的问题都解释的很清楚明白，但是其中对非扩展性锁——ticket lock 在 Linux 内核上的运行测试不够全面具体。作者提出了四个基准 FOPS、MEMPOP、PFIND、EXIM——前两个是微测试基准，后两个代表应用程序工作负载——从这四个方面测试了 ticket 锁在一个 48 核 2.6.39 版本 Linux 内核的机器上执行性能。我觉得可以再增添几个测试基准让应用程序工作方式更多样化，而且也应该在不同的机器上——核数不同或内核版本不同等等——来测试 ticket 锁的工作情况，这样可以得到更有说服力的测试结果。之后，作者提出了可以解释非扩展性锁行为的基于 Markov 的模型，并通过比较单核上微基准测试得到的实际速度与模型的预测速度来验证模型，此处测试数据不够充分。而且模型得到的结果也与实际运行结果有些微差别，主要是因为模型关注的是平均性能，没能充分预测运行时出现的不稳定性，我认为可以在下一步工作使此模型更加完善。然后作者介绍了几种可扩展性锁并给出了它们的应用性能测试结果，选择用 MCS 锁替代 Linux 中的 ticket 锁，这几种锁其实在实际应用中扩展性和性能都没有太大优势差别，为什么作者选择了 MCS 锁？文中并没有给出足够充分的原因，我认为作者应该多给出几种可扩展性锁在 Linux 内核的实现，然后比较它们与 ticket 锁的性能，从而确定选择哪种锁来替代 ticket 锁。