**文献综述**

学院：计算机科学与技术 班级：硕1708班 学号：M201773207 姓名：段沛云

# 前言

虚拟化技术通过在虚拟机之间复用物理资源，为云环境带来服务器整合的优势。多核处理器的出现也进一步扩大了虚拟化技术的优势。在基于多核物理平台的虚拟机系统中，虚拟机中的CPU(vCPU)[1]可以分时复用所有的物理处理单元，也可以限制访问部分指定的物理单元，这给虚拟机的资源分配带来更大的灵活性。而现代操作系统为了支持多处理器对内核进行了多方面的优化。比如为了保护共享数据会需要一些同步机制，但同时也会因为虚拟化技术而产生较大开销。

Read-copy update(RCU)是Linux内核中可实现的一种可扩展的高性能数据同步的方式，RCU的独特性包括支持并发读写，以及高度优化的CPU同步。从2002年RCU被引入Linux内核以来，其利用率不断扩大。对于被RCU保护的共享数据结构，读者不需要获得任何锁就可以访问它，但写者在访问它时首先拷贝一个副本，然后对副本进行修改，最后使用一个回调（callback）机制在适当的时机把指向原来数据的指针重新指向新的被修改的数据。显然这样的机制对于读者来说几乎没有同步开销。但是另一方面，由于RCU不需要锁，不适用原子指令，那么多年来对于一般锁机制的资源抢占而产生的开销分析并不能适用于RCU。

# 主体

一般来说，传统的多个处理器并行运行，通常需要通过信号量或者自旋锁保证临界资源的正确性。而在这样的前提下，自旋锁虽然能够对临界资源进行很好的保护，但是实际CPU处于请求锁的自旋时实际是在做无用功。一个处理器拥有自旋锁的时间越长，在等待中的vCPU自旋等待锁释放的时间就越长。不仅严重浪费CPU周期，也会导致优先级高的进程得不到调度。如何减小自旋锁开销增大的问题是当前虚拟机vCPU研究者关注的一个问题。

相关研究人员还提出了一些对自旋锁的改进方案，以适用不同的应用场景， 如读写锁、顺序锁等。顺序锁为写者赋予了较高的优先级，即在有读者访问资源的时候依然允许写者继续进行。考虑到原子操作的高效性基于 CAS ( Compare and Swap) 原语的无锁编程技术[2]也是时下的研究热点，不过无锁编程技术需要解决垃圾回收和 ABA 问题，为系统带来额外开销，而且基于CAS 设计的无锁数据结构通常应用场景较小，使用较为困难。另外有研究人员基于对linux调度器的分析，提出三种不同层次的免锁优化[3]方法: 基础优化、调度行为优化、基于上层应用特征的参数调优。分别从代码级、进程级和上层应用三个方面对自旋锁进行改进。除此之外，也有研究人员提出名为APPLES的框架[4]来解决这个问题。APPLES可以监视由于vCPU过度旋转或者抢占锁而引起的开销，并定期调整旋转阈值来减少开销。另外提出了用于虚拟机化环境的I-SpinLock机制[5]，该机制的原则是只有当vCPU的剩余时间片足够进入和离开关键部分的时候才允许线程请求锁。这种方法在多核情况下表现更好。

RCU机制也是*Paul McKenney*[6]为了解决自旋锁的问题而提出的，主要应用于读密集型数据结构中。尽管RCU机制很早就被提出，但在Linux内核中直到2.5版本才被采用。最初的RCU锁比较粗糙，仅仅是在读者读数据时禁用抢占，也就是在一个CPU上禁止切换进程。这就会产生一个问题：只要读者加锁就意味着禁用抢占，这就会很大的降低系统的交互性能，也会降低实时任务的性能。于是就出现了可抢占RCU，它引入了新的数据结构和逻辑控制机制。但是同样，为了保证读取的一致性，RCU在特定状态之间转换数据结构仍然需要等待所有现有读取的完成，而其中的等待时间就成为了瓶颈问题。文章[7]对RCU进行变形，提出了可在需要高吞吐量和可扩展更新的并行数据结构中使用的Predicate RCU（PRCU），该机制允许更新操作只需等待影响数据一致性的读取操作，极大地降低了更新开销。另一方面，随着Linux内核越来越复杂，在多核和低内存条件下实现RCU变得困难。文章[8]第一次在低内存模型下，使用最新开发的用于C/C++11内存模型GPS，实现验证了RCU机制。同样的，文章[9]也提出了一种在顺序一致性和低内存条件下测试Linux内核RCU主要代码的方法。

# 总结

虚拟机中资源分配问题大多数使用信号量和锁进行解决，无论是自旋锁还是顺序锁都会涉及到关于锁的抢占问题和对CPU时间片的浪费问题。现已有很多研究对该问题提出了不同的解决方案。RCU作为一种较新的锁机制，虽然较好的解决了锁抢占问题，但同时也可能带来了诸如峰值延迟，增加内存占用和内存碎片，降低虚拟机性能等等问题。

# 参考文献

1. 金海, 钟阿林, 吴松, 等. 多核环境下虚拟机 VCPU 调度研究: 问题与挑战[J]. 计算机研究与发展, 2011, 48(7): 1216-1224.
2. Peng Jian-zhang ,Gu Nai-jie,Zhang Xu,et al． Fast epoch: a fast memory reclamation algorithm for lock-free programming［J］．Journal of Chinese Computer Systems，2013，34( 12) : 2691-2695
3. 张旭, 顾乃杰, 苏俊杰. Linux 调度器免锁优化方法研究[J]. 小型微型计算机系统, 2017, 38(4): 690-695.
4. Shan J, Ding X, Gehani N. APPLES: Efficiently Handling Spin-lock Synchronization on Virtualized Platforms[J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(7): 1811-1824.
5. Teabe B, Nitu V, Tchana A, et al. The lock holder and the lock waiter pre-emption problems: nip them in the bud using informed spinlocks (I-Spinlock)[C]//Proceedings of the Twelfth European Conference on Computer Systems. ACM, 2017: 286-297.
6. P. E. McKenney and J. D. Slingwine. Read-copy update: Using execution history to solve concurrency problems. In *PDCS*, 1998.
7. Arbel M, Morrison A. Predicate RCU: an RCU for scalable concurrent updates[C]//ACM SIGPLAN Notices. ACM, 2015, 50(8): 21-30.
8. Tassarotti J, Dreyer D, Vafeiadis V. Verifying read-copy-update in a logic for weak memory[C]//ACM SIGPLAN Notices. ACM, 2015, 50(6): 110-120.
9. Kokologiannakis M, Sagonas K. Stateless model checking of the Linux kernel's hierarchical read-copy-update (tree RCU)[C]//Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software. ACM, 2017: 172-181.