可扩展可交换法则: 为多核处理器设计可扩展的软件

The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors

背景介绍

随着多核处理器称为趋势，操作系统的可扩展性变得日益重要。然而，操作系统的可扩展性不仅有其实现方式上的限制，也有接口定义的限制。好的接口定义可以给它的实现提供更多可扩展的机会。该论文提出，如果对于多个线程的调用，返回值和最终状态与这些调用的实际执行顺序无关，那么这样的接口是可交换的。可交换的接口可以用无冲突的方式实现，即不同线程所访问的系统状态的部分之间没有读写冲突，也就可以以可扩展的方式实现。

为了确保实际系统接口的可交换性，作者开发了commuter工具。该工具可以通过形式化的静态分析验证各个接口之间的可交换性，并通过生成测试调用序列来测试可交换性。另外，为了保证实现的无冲突，commuter工具还可以生成测试调用序列来测试实现的内存访问冲突。

根据可交换性原理，作者在xv6的基础上开发了sv6操作系统，使得该系统的由18个系统调用生成的26238个调用序列中，无冲突的测试样例个数相比Linux的 17206个提高到了26115个。在sv6系统中实现了新的文件系统ScaleFS、新的虚拟内存管理系统RadixVM和新的引用计数机制Refcache。将sv6系统运行在8块2.4GHz 10-core Intel E7-8870的机器上，并测试statbench、openbench和邮件服务器三个程序，都获得了显著的可扩展性提升。

可扩展可交换法则

多核处理器在并行运行时，若没有加任何同步机制，不同核的执行是完全异步的，在设计程序时不能对其先后顺序做任何假设；但是这对实现实际的操作系统接口来说很难做到，不同核上的系统调用，运行的先后顺序会影响调用的返回值或系统最终状态。如果反过来思考，如果不同核上的系统调用运行的先后顺序不影响调用的返回值或系统最终状态，是否可以不需要任何同步机制就能实现呢？

SIM-可交换性

将一次系统调用表示成调用请求和得到回应（返回值）两个事件，分别表示为和，并用发起调用的线程编号标记，如和分别是1号线程调用A和2号线程调用B。用这样的符号可以表示一个系统的历史记录，比如

一个历史记录的子历史记录为其中某个现程的调用序列，如

如果一个历史记录满足对任意线程都有

则称是的一个重排。注意重排只是不同线程之间的顺序重排，同一个先内部的多个调用之间的顺序不变。

状态依赖的接口基的可交换性（state-dependent interface-basis commutativity, SI-commutativity）的定义为，在给定的系统初始状态下，某个历史记录与其重排所产生的结果都符合接口定义。进一步，状态依赖的接口基的单调可交换性（state-dependent interface-basis monotonic commutativity, SIM-commutativity）的定义为，对一个由调用序列组成的历史记录，如果该序列的任何前缀组成的历史记录都是SI-可交换的，则称是SIM-可交换的。

可扩展可交换法则

如果一个实现满足接口规范，则称该实现是正确的。如果实现的两个步骤被不同线程调用时，一个线程写系统状态的某个成分，同时还有其它线程在访问系统状态的同一个成分，则称为实现是有冲突的；反之则称实现是无冲突的。

可扩展可交换法则（scalable commutativity rule）：假设一个接口规范有一个正确的参考实现，考虑一个历史记录，其中是SIM-可交换的，且能产生，那么存在一个满足规范的正确实现，且在执行中的调用序列时，在部分是无冲突的。

假设已有了参考实现，且的状态为。考虑一个针对中这个特定调用序列而设计的实现。初始时系统在回放模式，当的接口被调用时，若是严格按照中的调用序列来调用的，则直接查找历史记录中的返回值作为该次调用返回值；若第一次发现一个调用和调用序列不符，则计算出当前的，并进入仿真模式，之后的所有调用都通过直接调用来实现。当前系统处于回放模式还是仿真模式由状态决定。

注意到，如果调用序列就是中的调用序列，那么系统将始终处在回放模式，而不会进入仿真模式。这样的实现在这个调用序列下仍然是不可扩展的，因为多个线程要同时访问的调用序列，并更新所期望的下一个调用。

在的基础上做修改即可构造一个可扩展的实现。由于可以重排而不影响结果，那么系统状态只需要保存的子历史记录即可，这些子历史记录任意组合成的调用序列都能和产生同样的结果。于是可以构造：初始时系统在回放模式，当的接口被线程调用时，若是严格按照的调用序列来调用的，则直接查找历史记录中的返回值作为该次调用返回值；若第一次发现一个调用和调用序列不符，则计算出当前的，并进入仿真模式，之后的所有调用都通过直接调用来实现。

同样，如果调用序列就是中的调用序列，那么系统将始终处在回放模式，而不会进入仿真模式。而在回放模式下，不会被写，不会被访问，而每个线程都只对各自的进行更新操作，所以调用都是无冲突的。

这样就构造了一个在的调用序列下无冲突的实现。值得注意的是，这个实现在遇到其他调用序列不是无冲突的。也就是说，仍然不能指导实际系统的具体实现，因为在实际系统中可能的调用序列数量非常多，针对某个特定的调用序列做出的优化非常不值得。

无冲突性与可扩展性

在现代多核处理器上，有冲突的实现需要占用核间通信网络来维护缓存一致性，往往因此而变得不可扩展。无冲突的实现通常是可扩展的，但也有例外；对于某些访存密集型程序，即使在多核运行时不需要核间通信网络来维护缓存一致性，但多个核仍然会竞争主存的访问带宽，使得性能受限与主存访问带宽，而不是随着核数的增多而扩展。

分析接口的可交换性

为了分析接口的可交换性，作者开发了commuter工具。commuter工具由三个部分组成：analyzer、testgen和mtrace。analyzer是验证和测试系统接口可交换性的工具，testgen是测试用例的生成工具，而mtrace是测试系统接口的实现的无冲突性的工具。

analyzer用两种方法来分析接口的可交换性：具体测试和符号化验证。在具体测试中，对一个生成的初始状态和调用序列，该工具取调用序列的所有前缀，对每个前缀生成所有可能的重排，通过实际运行得到返回值和最终状态，然后检验这些值是否相等。

在符号化验证中，需要先对接口规范建模，然后进行符号化执行。例如在验证接口和接口的可交换性时，通过符号化地先运行再运行得到返回值和，以及最终状态，然后调换顺序运行得到返回值和rBA，以及最终状态，然后用自动化证明来证明、和在符号化执行的过程中，每遇到分支则需要分别执行两条路径，若有条执行路径，有条执行路径，则执行两个调用共有条路径，要证明两次执行结果相等要考虑种情况。

testgen在生成测试用例时，不仅需要考虑路径覆盖率，还要考虑冲突覆盖率。路径覆盖率要求尽可能覆盖代码中的每一条执行路径，而冲突覆盖率在要求尽可能覆盖共享数据结构上的每一种访问模式。

mtrace在虚拟环境中运行整个操作系统，并保存所有的内存访问记录，通过这些记录可以测试接口实现的有无冲突。

设计可扩展的操作系统

作者设计了sv6操作系统，尽可能使接口定义可交换，尽可能使接口实现无冲突。用commuter工具分析可发现，该系统的由18个系统调用生成的26238个调用序列中，无冲突的测试样例个数达26115个，相比之下Linux系统只有17206个。在sv6系统中实现了新的文件系统ScaleFS、新的虚拟内存管理系统RadixVM和新的引用计数机制Refcache，使得可扩展性大幅提升。

除了使用per-core数据结构、RCU数据结构、可扩展的锁等技术外，在可扩展可交换法则的指导下，sv6的设计遵循了这些思想：

1. 使用可交换的数据结构，如数组、基数树、哈希表等。某些数据结构是不可交换的，比如平衡树，树节点的插入顺序不同，得到的树的结构也不同；事实上，平衡树的插入和删除操作很容易产生冲突，因为一个节点的插入与删除需要涉及到对其它节点的操作。
2. 延迟对时间不敏感的操作。比如在新的引用计数机制Refcache中，每个核拥有自己的一个计数器，加一和减一只在自己的计数器上进行，这样一来判断计数器是否为零仍然是一个有冲突的操作。当一个核执行了减一的操作后，它不会立即检查计数器是否为零。因为资源释放对时间不敏感，可以每隔一段时间进行一次垃圾收集，判断各个资源的引用计数是否为零，并释放掉引用计数为零的资源。这样当一个资源有较多引用者的时候，判断是否为零的操作就会显著地减少。
3. 避免不必要的读。比如当用户只想知道某几个挂起的信号，而POSIX的sigpending调用却返回了所有的挂起信号，这就容易产生冲突。

实验

将sv6系统运行在8块2.4GHz 10-core Intel E7-8870的机器上，并测试statbench、openbench和邮件服务器三个程序。

测试了statbench 的两个版本，一个调用了fstat，另一个调用了一个新的接口fstatx，而fstatx和link、unlink是可交换的。根据测试结果，用Refstate计数机制实现fstat比原版的fstat还要慢，但是这种情况下并发的link和unlink却相比原版获得了数倍的加速。但是，用Refstate实现的fstatx才是真正可扩展的，其性能比fstat的两种实现都要高，且能完美地扩展到80核。

openbench测试的是open调用。由于POSIX中“最小可用FD”这一规定使得open与自身不可交换，sv6提供了一个额外的标志位，可用于指明是否需要满足“最小可用FD”这一规定，还是选用任意可用FD。但实际测试结果表明，不管使不使用这一标志位，sv6上的openbench性能都能较好地扩展，相比Linux有很好的加速。

邮件服务器是一种实际应用，它涉及的系统调用除了fstat和open以外，还有创建新进程、网络相关的消息队列等。在sv6中，创建新进程不再通过fork/exec的方式，而是直接创建新进程，不复制当前进程；而消息队列也除了传统的有序模式以外，还提供无序模式，每个核有各自的消息队列，只有当一个核尝试从空队列中取出消息时，为了负载均衡，它需要从其他核的队列中取。测试结果表明，在80核上的性能相比10核有7.5的加速比。

结论

我们正处在软件性能快速转变的时代，为了性能必须引入并行。该论文为可扩展软件的设计、实现和测试提供了一种新的方法。通过定义SIM-可交换性，提出了可扩展可交换法则。进一步，开发了commuter工具，能从接口模型和实现两方面测试可交换性，同时还能测试实现的无冲突性。最后，开发了sv6操作系统，用可扩展的方式实现了POSIX的大部分接口，定义了一部分新接口来满足可交换律，并用可扩展的方式实现了这些接口。随着可扩展性在软件性能中成为越来越重要的因素，可扩展可交换法则将会给开发者提供更多的指导，以应对这一挑战。