The Scalable Commutativity Rule:Designing Scalable Software for Multicore Processors

`Scalability` `Commutativity` `Multicore`

Clements, A. T., Kaashoek, M. F., Zeldovich, N., Morris, R. T., & Kohler, E. (2015). The scalable commutativity rule: Designing scalable software for multicore processors. ACM Transactions on Computer Systems (TOCS), 32(4), 10.

背景

接口实现之前，是否能够通过接口的逻辑详述来确定其可扩展性的情况？本文提出一个规则： 当一个接口的操作存在交换性时，那么总存在一种具有较好可扩展性实现方式。本文基于此，提供了一套工具，这套工具能够对高级的接口模型进行分析，产生大量的测试案例，对接口实现的可扩展性进行评估。

可扩展性、接口、交换性

一般方法是：在不同核数的条件下选择负载进行系统的可扩展性测试，结合一些工具如differential profiling[29]来找出影响扩展性的瓶颈。但这种方法很可能找不到根本的影响因素。
COMMUTER 基于一种简单化、符号化的接口模型，计算特定环境下哪些操作具有交换性，并对一种无冲突的实现进行测试。

- 可扩展性。对共享内存的互斥操作会带来扩展性的问题。一些情况下，内核并不需要线性一致性，即顺序化（The Law of Order）。
- 接口。传统的方法遵循：设计、实现、测试、重复这样的循环，仍然有没有找出的可扩展性的瓶颈，并且不能解释这些瓶颈是不是因为系统调用造成的。
- 交换性。一些工作利用交换性来考虑并行操作的安全性，这里则考虑可扩展性。在这里，交换性与不冲突的共享内存访问可以进行对应。SIM Communitativity: state-indepent,interface-based,monotonic。

The Scalable community rule及其证明

系统的执行建模为一系列的操作，这些操作为触发（invocation）或回应(response)。一个系统执行历史可以表示为：

$$H = A_1\ B_1\ C_2\ A_2\ C_1\ B_2\ D_2\ D_1\ E_1\ E_2\ F_1\ G_2\ H_2\ F_2\ H_1\ G_1$$

$$H|_t = \boxed{A}\boxed{A}\boxed{D}\boxed{D}\boxed{H}\boxed{H}$$

- SI-Commutes，Y本身可能不具有单调性，因而下面的规则只能表明SI-commutes.

$$Y\ SI-commutes\ in\ X||Y \coloneqq \forall Y' \in reorderings(Y), Z : X||Y||Z \in \mathscr{S} \Leftrightarrow X||Y'||Z \in \mathscr{S}.$$

- SIM-Commutes

$$Y\ SIM-commutes\ in\ X||Y \coloneqq \forall P \in prefixes(reorderings(Y)) : PSI-commutes\ in\ X||P.$$

Commuter架构及其实现

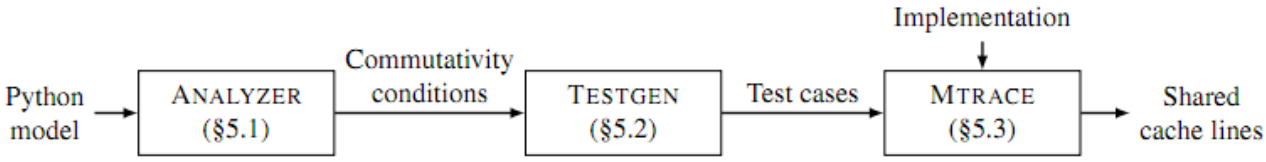

Figure 3: The components of COMMUTER.

- 分析器（Analyzer）。对接口的交换性质进行分析。输入是接口的行为模型（Python）,分析器会对所有可能的操作组合进行分析，输出接口sim的条件。具体可参考rename(a,b)和rename(c,d)的例子，以及Python代码
- 测试样例生成。将分析器的sim条件转换成具体的测试样例。一方面需要覆盖所有的执行路径，另一方面，还需要覆盖一些冲突的情况，如在相同的执行代码路径下，两个pwrites属于不同的访问模式。访问模式？ 不同的同构的组？
- Mtrace. 在实际的实现中对测试样例进行检测，如果存在冲突，则报告共享的变量和访问他们的代码。

测试与结果

对Linux和sv6两个操作系统进行了测试，实验结果表明了本文给出的规则的可用性。

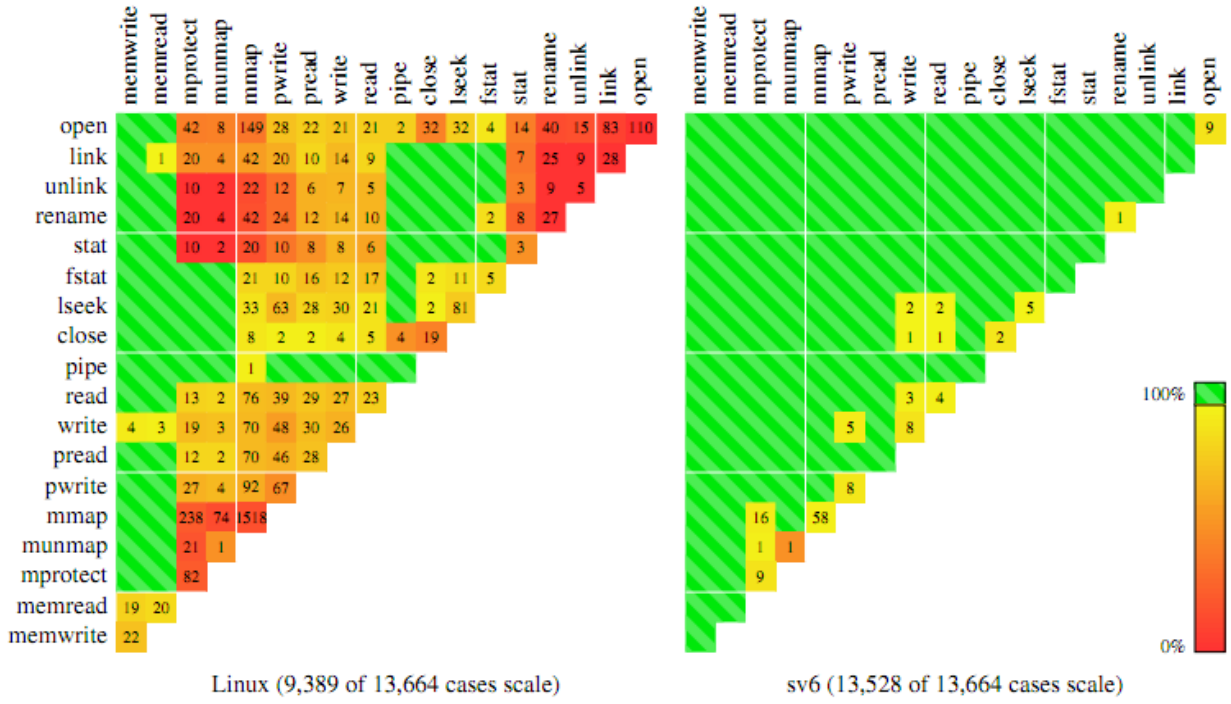

Figure 6: Scalability for system call pairs, showing the fraction and number of test cases generated by COMMUTER that are not conflict-free for each system call pair. One example test case was shown in Figure 5.