可扩展交换规则：为多核处理器设计可扩展软件

这篇文章来自麻省理工 CSAIL 实验室下的 PDOS 小组，本文考察了 interface(如 system call)对软件扩展性的影响。 核心问题是, 如果把接口和实现良好的分离开来, 那接口的设计中是不是就蕴含了一些天然的限制, 使得无论怎样实现, 都无法突破这些天然限制.。

这篇论文共有 8 小节，主要内容放在 3-7 章节。前 2 章介绍了一些现状以及提出了一些想法，并对此展开深入实现，然后简要概括了相关工作。

第三部分提出了可伸缩交换性规则的思想，如果它们的实现具有无冲突内存访问，则声明一组操作规模，并且通信可以帮助避免无序操作之间的内存访问冲突。因此，如果这样的接口操作可以通勤，可以实现可扩展性。在共享内存系统中，如果一个内核没有写入由另一个内核读取/写入的高速缓存行，则操作会通信。 在这种无冲突的情况下，接口操作以可扩展的方式进行通信。 正式的 SIM 交换性是一种依赖于状态的（S），基于接口的（I）和单调（M）属性，它捕捉了如果操作顺序与未来操作无法区分就无关紧要的想法。 在 SIM 交换的假设下，存在一个无冲突的正确实现。 为了利用这个规则，现有的 POSIX 接口必须分解复合操作，允许非决定性和弱排序，并且异步释放资源。 这涉及修改现有的接口语义。

SIM-交换性规则是论文的主要贡献之一。 作者发现交换性的简单代数定义过于严格。 相反，他们创建了一个定义，说明如果一个历史 Y 在 H 中交换，存在一个正确的实现，其中区域 Y 是无冲突的。 这意味着内存访问是缓存行的互斥区域，这意味着该接口会缩放。SIM 交换特别意味着操作必须是状态依赖的，基于接口的和单调的。 本规范的细节留给论文。 在一般水平上，如果操作可以以任何方式重新排列以产生相同的结果，则动作序列 SIM 通勤（并且随后缩放）。

在第四部分，作者提供了设计交换界面的建议。 首先是分解复合操作。 POSIX API 通常包含几个不通勤的子操作。 如果可能的话，这些操作应该分开。其次是拥抱规范非确定性。 这允许更少的同步和更多的并发性。 第三，许多接口需要严格的排序，当仅需要弱排序时，这限制了交换性。 最后，异步释放资源意味着可以放弃昂贵的同步，并且可以再次改进交换性。

第五部分，作者编写了一个称为 COMMUTER 的工具, 一种在实际实现创建之前使用高级接口模型分析来设计可伸缩软件接口的工具，它具有测试驱动的方

法。COMMUTER 使用另一种名为 ANALYZER 的工具，它查看使现有接口可交换和可扩展所需的路径和冲突条件。如果没有 COMMUTER，确定接口的可扩展性并改进它的常见方法是在不同数量的内核上使用给定的工作负载对其进行测试，然后使用差异分析等工具查找瓶颈并在重新测试之前对其进行改进。 作者使用 COMMUTER 开发了名为 sv6 的实验操作系统，该系统改进了 POSIX 软件接口调用中的 18 个，以提高可伸缩性。实验表明对 COMMUTER 产生的 13664 个 test 而言, linux 能对其中的 68%良好扩展, 而 SV6 能良好扩展其中的 99%。

在论文最后部分，作者给出了详细的评估。 在 COMMUTER 中对几个 POSIX 文件系统和虚拟内存调用进行建模后，用于评估 Linux 的可伸缩性。 将 18 个 POSIX API 与 sv6 API 进行比较。C OMMUTER 确定 sv6 在 13,664 次测试中的 13,528 次是无冲突的，而 Linux 在 9,389 次测试中没有冲突。 为了进一步确认真实硬件的可扩展性，使用一些微型基准和邮件服务器应用程序进行评估。 使用 statbench，比较链接/取消链接与统计。 Openbench 用于检查文件打开操作和文件描述符分配的可伸缩性。 在这两个微基准中，交换版本都按照它们的非交换对象进行缩放。 邮件服务器有很多混合的操作，非交换操作会导致基准测试的吞吐量崩溃，但交换配置合理扩展。

本文提供了一个非常好的交换性观点，并提供了现有接口的具体示例来帮助理解问题和解决方案。 这些例子使得理解这些概念变得更容易。 此外，性能评估清楚地表明交换性有助于实现可扩展性。