阅读报告

文章主要内容

文章：mTCP: A highly scalable user-level TCP Stack for Multicore systems 文章发表于 2014 年的 NSDI,这是一个 CCF B 类的会议，网络方向。

扩展提高在多核系统中短的 TCP 连接的性能很具有挑战性。尽管有很多在这方面的提议来改进各种缺点，但是低效的内核实现问题依然存在。本文提出了 mTCP,一个高性能的用户级别的 TCP 协议栈，用于多核系统。MTCP 从底向上解决低效问题—从包 IO，TCP 连接管理到应用接口。主要的工作：1）将多个昂贵的系统调用修改为单个共享内存引用。2）允许高效的流级别的事件聚合。3）实现批处理的包 IO。

小的 TCP 流占据所有 TCP 连接中的大多数，尽管当前取得了一些进展，linux TCP 交互速率高峰时达到每秒 30 万个交互，数据包的 IO 达到每秒千万级别。

之前的工作将原因归结为 2 大类：操作系统中系统调用的高开销和多核系统上造成资源争用的低效实现。针对前者，需要做大量内核的修改的 IO 抽象，需要大量修改应用程序。针对后者，主要在已有的实现上做一些增量式的变化，因此并不能完全解决低效性。

本文就是寻找一种提供高性能但是又不需要大量修改已有代码的方法。本文设计了一个用户层的 TCP 协议栈，利用了高性能的包 IO 库，这些 IO 库允许应用直接访问数据包。

用户态的协议栈的设计目标：TCP 协议栈的多核扩展性；容易使用；容易部署。

内核态的 TCP 协议栈的限制：有很多研究工作旨在解决 linux TCP 协议栈中的 4 个主要的低效：连接本地行的缺乏，共享的文件描述符空间，低效的包处理和沉重的系统调用负载。下图是经典 linux 和一些已有工作在各方面性能上的比较。

	Accept queue	Conn. Locality	Socket API	Event Handling	Packet I/O	Application Mod-ification	Kernel Modification
PSIO [12], DPDK [4], PF_RING [7], netmap [21]	No TCP stack				**Batched**	No interface for transport layer	**No (NIC driver)**
Linux-2.6	Shared	None	BSD socket	Syscalls	Per packet	**Transparent**	**No**
Linux-3.9	**Per-core**	None	BSD socket	Syscalls	Per packet	Add option SO_REUSEPORT	**No**
Affinity-Accept [37]	**Per-core**	**Yes**	BSD socket	Syscalls	Per packet	**Transparent**	Yes
MegaPipe [28]	**Per-core**	**Yes**	lwsocket	**Batched syscalls**	Per packet	Event model to completion I/O	Yes
FlexSC [40], VOS [43]	Shared	None	BSD socket	**Batched syscalls**	Per packet	Change to use new API	Yes
mTCP	**Per-core**	**Yes**	**User-level socket**	**Batched function calls**	**Batched**	Socket API to mTCP API	**No (NIC driver)**

没有任何一个单独的系统能够在这 4 方面都做到不错的性能。

为什么使用用户态的 TCP：尽管有很多工作尝试扩展多核系统中的 TCP 的性能，他们之中很少可以克服上述的内核的各种不足。即使是当前最好的工作 MegaPipe 在内核上花费的 CPU cycle 也是非常多的。但是这些 CPU circle 并没有得到很好的利用，处理相同的数量的 TCP transactions，linux 是 mTCP 花费时间的 4 倍。作者通过一个小实验，在处理大量并发的 TCP 连接的时候，观测 CPU 的使用率。实验结果表明：linux 和 MegaPipe 花费了 80%到 83%的 CPU circles 在内核上，将较少的 cycles 留给用户应用。

至此，工作的动机比较明确：设计一个用户态的 TCP 协议栈，融合所有的这些存在的

优点成为一个系统, 这样的一个系统能够获得多大的收益？这样的设计可以将已有的数据包的 IO 库的性能优化带到 TCP 栈吗？用户态的 TCP 协议栈有这样几个吸引人的地方：1.剥离了内核的复杂性，开发难度降低。2.用户态的 TCP 协议栈可以很好地利用已有的一些数据包库的优化来提升性能。3.可以有效实现批处理，除了对数据包的 IO 实现批处理之外，用户态的 TCP 可以从应用中收集多个流级别的事件，而无需频繁的模式切换。4.这可以使我们很容易地保留已有应用的编程接口。本文实现的 TCP 协议栈是向后兼容的，提供了一个 BSD-like 的 socket 接口。

设计 :首先要清晰系统的设计目标 :mTCP 的目标就是在多核系统上实现较高的扩展性，同时对存在的多线程，时间驱动的应用保留向后兼容性。在高层上，应用链接到 mTCP 库，位于底层的 2 个组件是：用户态的 TCP 栈和数据包 IO 库，负责实现高扩展性。用户态的 TCP 实现在每个 CPU 核上作为一个位于同一个应用进程中的线程运行。mTCP 线程使用定制化的数据包 IO 库来直接传输和接收来自网卡上的数据包。已有的用户态的数据包库只允许一个应用访问一个 NIC 端口。因此 mTCP 每个网卡端口上只支持一个应用。这一瓶颈在将来的虚拟化技术中能够得到解决。同时我们的设计还能够保证已有的应用继续使用存在的 TCP 栈，只要这些应用只使用不被 mTCP 使用的网卡即可。

下面分别介绍底层的 2 个组件：用户态的数据包 IO 库：用户态的 TCP 栈。

用户态的数据包 IO 库：已有的一些库并不适合实现传输层，因为他们的接口主要基于轮询。轮询会浪费大量的 CPU cycles。我们的系统需要实现在多个网卡上的 TX 和 RX 队列实现高效的复用。例如，我们不想 block 一个 Tx 队列，在发送一个数据包的时候一个控制包(SYN 和 ACK 等)在等待被接收。为了解决这些挑战，mTCP 对 PSIO 实现了扩展来实现高效的事件驱动的数据包 IO 接口。在 PSIO 的高速的数据包 IO 的基础上，新的事件驱动的接口允许 mTCP 线程高效地等待来自 RX 和 TX 队列的事件。PSIO 的使用使得可以将系统调用和上下文切换的负载摊销在整个系统上, 以及消除每个数据包的内存分配和 DMA 开销。

用户态 TCP 栈：一个用户态的 TCP 协议栈消除了很多的系统调用，这可以显著的减少 linux 内核中 TCP 的负载。实现用户态的 TCP 栈的一个方法是：将其完全作为一个库来实现，并作为应用主线程的一部分来运行。这种 0 线程 TCP 可以提供最好的性能，因为这将昂贵的系统调用转化为轻量级的用户态的函数调用。然而这种方法的限制是 : 内部 TCP 处理的正确性依赖于应用中 TCP 函数的及时调用。

在 MTCP 中，我们选择创建一个独立的 TCP 线程来避免这样的问题，以及减少移植应用的开销。文章随后从基本 TCP 处理，无锁，每个核的数据结构，批处理化的事件处理，短连接优化等方面介绍 mTCP 如何提高性能。

实验的评价主要是 2 个方面：多核可扩展性，移植应用上的性能提升。随着核数的增多, mTCP 每秒处理的 transaction 的数量逐渐增加, 比 Linux, REUSEPORT 和 MegaPORT 要好。对于后者作者主要做了 web server 和 SSL 代理的实验。

论文有趣的地方

发现论文的作者还有其他的研究兴趣：Emerald programming language,这是一个分布式的面向对象的编程语言。最近我也在看一些区块链的文献，二者的结合？