Scalable Kernel TCP Design and Implementation for Short-Lived Connections阅读报告

这是一篇发表在ASPLOS 2016的文章，合作单位有新浪，知乎和清华大学。会议全称是：Architectural Support for Programming Languages and Operating Systems，是CCF中计算机体系结构/并行与分布计算/存储系统中的A会

文章内容

带宽增加，单机核数增加以及应用对短连接需求的增多，对高性能的和可扩展的TCP协议栈的需求增加。尽管有很多全新的方案提出，但是在产业界我们依然需要自底向上并行的TCP协议栈并且能够兼容已有的应用。本文提出了兼容BSD SOCKET的并且是可扩展的内核socket设计--Fastsocket。并且已经在新浪微博中部署。Fastsocket是一个自底向上是隔离设计的，这消除了在协议栈中的各种锁的争用。

背景

短的TCP连接数量的增多；TCP连接的建立和关闭成为性能的关键；多核平台的增多，使得网络栈的可扩展性成为重要提升性能的方法；短连接的特点：频繁的昂贵的TCP的建立和销毁，导致了在TCB和虚拟文件系统中严重的共享资源争用。 全局TCB在Linux内核中表示为TCP socket。所有的sockets使用2个全局哈希表进行管理--listen table和established table.因为这2个表在全系统内都是共享的，CPU核之间的同步不可避免。TCB的访问和更新是在2个阶段完成的：进入的数据包触发的core的中断；对进入的包进行进一步的处理以及对与出包的封装，这个是在运行对应应用的核中运行完成的。上述两个步骤发生的过程可能不在同一个核中。最好的情况是希望这2个阶段都发生在同一个核中，connection locality. VFS抽象：socket是由VFS抽象对用户应用开放，作为socket文件描述符。打开和关闭该文件描述符对TCP连接的建立和销毁的性能有重要影响。在处理VFS共享的状态时会有巨大的同步开销。 核数增多会带来锁的争用开销。 生产环境需求：

- 不同的TCP/IP实现上的兼容性
- 安全性
- 资源隔离和共享

贡献

- 介绍了新的内核网络协议栈设计，表级隔离，连接本地性，解决了网络处理path上的所有争用
- 阐明了BSD的API不一定是网络实现扩展性中的障碍。兼容性好，现有应用想要使用fastsocket无需修改
- 对内核做稍微的修改，可以构建扩展性很强的网络协议栈

挑战

- TCB管理

全局listen表

已有的解决方案：SO*REUSEPORT和Megapipe使用了socket层的划分来解决扩展性问题。在SO*REUSEPORT中，每个应用程序创建一个原来listen socket的拷贝，这些拷贝被链接成一个桶list.当处理新的连接请求时，内核遍历这个list,随机选择拷贝。应用程序只接受来自其自己的listen socket的请求。带来的缺点是：新的请求来的时候，必须遍历整个list来选择一个listen socket.随着核数增加，遍历寻找到一个匹配的listen socket的开销增加。

一个直白的想法就是表级的隔离，每个CPU核有自己的listen表，然而这样破坏了TCP的鲁棒性，一个核上的crash，即使其他核上有可用的进程可以用来处理用户请求。

全局estabished表

一个连接建立好之后，就被加入到这个表中。Linux当前采取每个桶一个锁的方式来同步对established表的多个核的并发访问及修改。这种锁的粒度已经不能在当前的生产环境中很好的工作。直接将该表分离不可行。因为有这种情况的可能：socket是由一个核创建的，并且插入到该核的本地表，然而该连接的后续数据包却有可能到达另外的核上。

- 连接本地性的缺乏

  为了尽最大可能地使用CPU的缓存，尽量将同一个连接的所有活动安排在一个核上。除了性能的考虑，如果可以确保一个连接的所有活动都在一个核内处理，全局established表的隔离就不是问题了。完整性是真正的挑战。已有的方案RFS(receive flow deliver)引入了一张表来追踪记录系统中的连接。

- 兼容性带来的额外负载

系统设计

目标：重新设计一个内核级别的TCP栈，保持对BSD socket API的兼容性同时实现高的可扩展性和低的负载。

主要有三点变化：

- 隔离全局共享的数据结构，listen table和established table
- 正确地导向进入的数据包以保持连接本地性
- 提供一种快速的路径，使得VFS中的socket能够解决扩展性问题，并且将该设计包装在BSD socket API中。

体系结构

由3部分组成：隔离的TCB数据结构（本地listen表和本地established表），receive flow deliver(RFD)和fastsocket感知的VFS

TCB管理的完全隔离

主要是对本地Listen表和本地established表的介绍

本地listen表：在内核TCP建立阶段，Fastsocket为每一个CPU核分配一个本地的listen表同时维护一个原来的全局listen表。进程调用local*listen()来通知内核其想在绑定的核上接收。当有连接进入，RSS将连接分配到核0上，内核搜索核0的本地listen表来寻求匹配。三次握手成功后，一个准备好的连接已经放入了本地listen socket的接收队列。当绑定在核心0上的进程接受了新的连接，核首先检查全局listen socket的接收队列。该操作无锁。然后检查核0的本地listen表寻找准备好的连接，并返回给进程。 本地established表：每个核心分配一个本地established表，新的建立完成的sockets插入到本地established表，在net*RX SoftIRQ中，内核检查本地established表来针对任意进入的数据包匹配established socket.

完全连接本地性

本部分主要介绍receive Flow deliver(RFD)来解决主动连接本地性问题。 Insight：当建立主动连接时，内核编码当前的CPU核ID为源端口。使用了一个hash函数，将端口映射为核的ID。RFD收到数据包时根据目的地端口的哈希决定导向到哪个核。

与此同时，在保持主动连接的本地性的同时，还需要继续保持被动连接的本地性。 RFD在对进入的包进行处理时，需要区分是主动连接的包还是被动连接的包，否则破坏被动连接的本地性。判断过程如下：检查进入的数据包，如果源端口是小于1024的，主动连接。若目的端口是2014以内的，则被动连接。若前2条都不满足，检查该包是否可以匹配到一个listen socket。匹配到则为被动连接。因为监听的端口不能同时发起主动连接。

利用硬件特征：我们的方案可以不需要任何硬件网卡的特征来支持，但是可以借助硬件特征做性能提升，如FDir来卸载一部分压力到网卡上。FDir的两种模式的选择：Perfect-filtering:全部放在网卡上，性能好但是需要额外的网卡配置；ATR模式：大部分的进入的包由NIC处理，剩下的由RFD处理。如何选择哈希函数：考虑负载和硬件能力。

fastsocket感知的VFS

在很多操作系统中，sockets是由VFS抽象管理的。然而sockets与磁盘上的文件有很大不同。

- socket都在内存中，不在磁盘上。从inode和dentry中无法获益。
- 编号定位socket不需要目录路径。

在socket的使用中，inode和dentry完全没有用上，但是为了和VFS的兼容，现有的实现中包含了。在频繁创建sockets时会带来巨大负载。

我们的方案：为socket的VFS处理提供fast path.在创建socket时跳过dentry和inode的管理。出于兼容性的考虑我们不能将dentry和inode直接移除。

评价

评价目标：Fastsocket确实能够对真实应用的性能有提升吗？相比于当前的Linux实现，Fastsocket是否对短连接的扩展能力很强。 8核 HAProxy-lood balancer 53.5% 24核 Nginx HAProxy 267% 621%

实验建立

3台机器，24核，网卡：RSS enabled,网卡队列数量根据核数来确定，配置网卡interrupt affinity和XPS(transmit packet steering),将每个RX和TX队列分给每个CPU核。因为baseline的内核使用单个listen socket不能充分利用CPU资源，所有的benchmarks监听在不同的IP但是在相同的80端口。

应用性能

生产环境评价：对照试验，两台机器，处理相同的请求数量，CPU利用率。 SLA：满足客户的延迟需求，生产环境中需要控制CPU利用率在某一阈值下，若某个CPU核达到了阈值，该核上的请求就会违背SLA，延迟不满足。因此在不同的核之间进行负载均衡是很重要的优化。 Fastsocket在核之间的负载均衡也比普通的Linux做得好。各个核上的负载方差小。

Nginx：FastSocket性能最好，吞吐能力最强；而且随着核数的增加，吞吐逐渐增加，不会像base一样反而会下降。

HAProxy:主要是测试主动连接的改善。Receive Flow Deliver提高了性能。

性能得到改进的原因的分析： * 锁争用：作者还做了锁争用的实验，使用的工具是lockstat;在enable fastsocket不同的组件的时候，对性能的改进也是不同的。同时还分析了各个锁的争用次数。 * 连接本地性：评估RFD的性能，分别打开或者关闭RFD，并且组合不同的网卡特征。评价指标是吞吐率和L3缓存的命中率。

相关工作

大量的工作集中于对网络密集型的应用提供系统支持；也有一些工作提出了全新的IO抽象，但是需要全部重写已有的应用，因为API的变化；还有设计全新的协议栈，使用LwIP和用户态的协议栈。虽然这些工作可以提高性能，但是不能复制所有内核的高级网络特性。这些高级网络特性往往是网络系统中使用最频繁的。

减少共享资源的争用

针对VFS：更细粒度的锁方案；绕过VFS； 针对listen socket:为每个CPU核拷贝listen socket.

实现连接本地性

批处理系统调用和0拷贝通信

批处理减少上下文切换的开销；

在语义上放松对系统调用的限制

很多研究认为是POSIX specification中的最低可用文件描述符限制了可扩展性。因此放松了这一rule.但是这样破坏了兼容性。

用户态的网络协议栈

应用特定的可扩展性是这类工作的最大动机。缺少BSD socket API而且实现的鲁棒性和一般性差。

带有硬件虚拟化的库操作系统

外展式核。库操作系统通常作为虚拟机运行。使用虚拟化隔离IO数据平面和控制平面。

为网络保留核

只有一个核来处理网络，减少争用。其他核跑应用。网络核容易过载。

结论

Fastsocket,新的扩展能力强的TCP socket，小流环境，隔离表，兼顾了连接本地性和兼容性。部署性好，已经在新浪得到部署。

一些想法

锁和隔离的开销。