腾讯云

zTCP-用户态TCP/IP协议栈

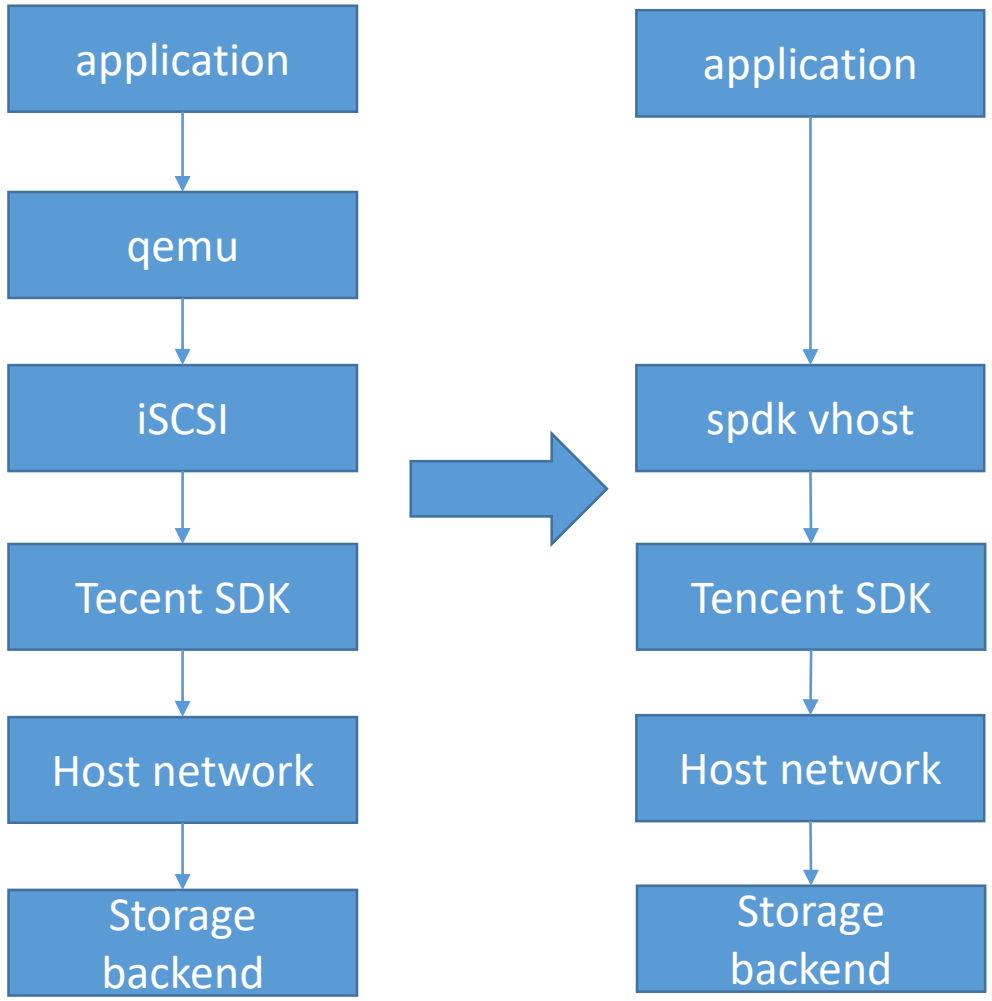
阎淼 腾讯云高级工程师

背景

- CBS块存储是虚拟化的核心服务
- 需要不断提升CBS块存储的性能

iscsi vs vhost-spdk

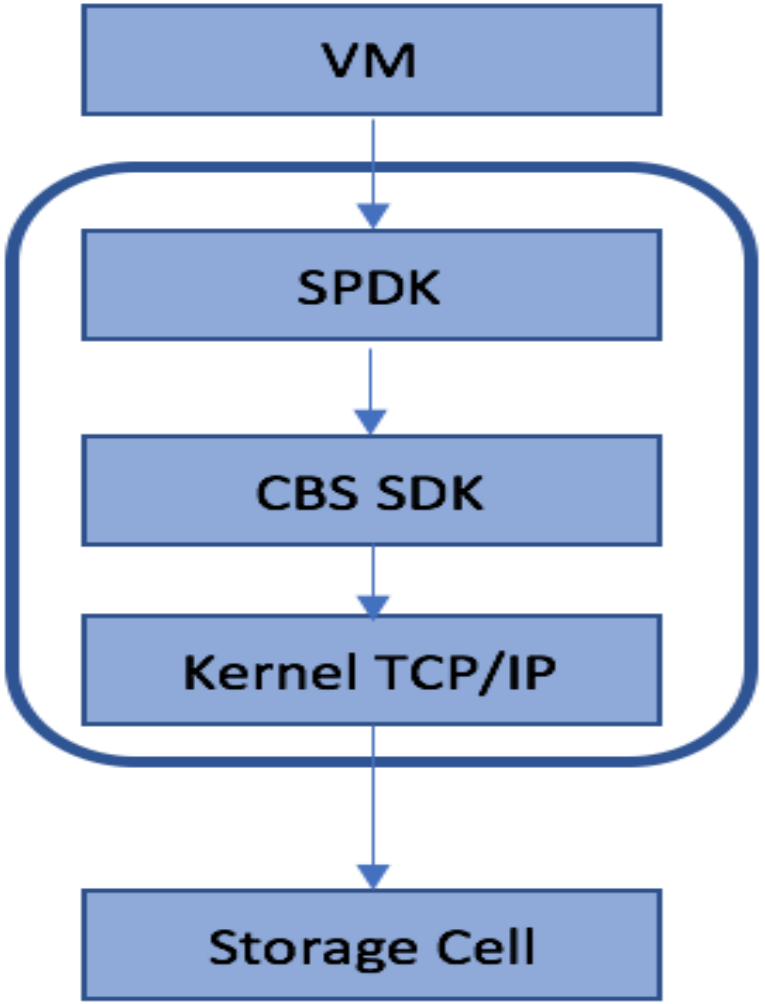
- 第一代采用qemu + iscsi架构
- 第二代采用spdk + vhost架构
- spdk对第一代架构的改善
  - 缩短了IO路径
  - 避免了频繁vmexit
  - spdk reactor轮询方式
  - 避免系统调用



性能分析

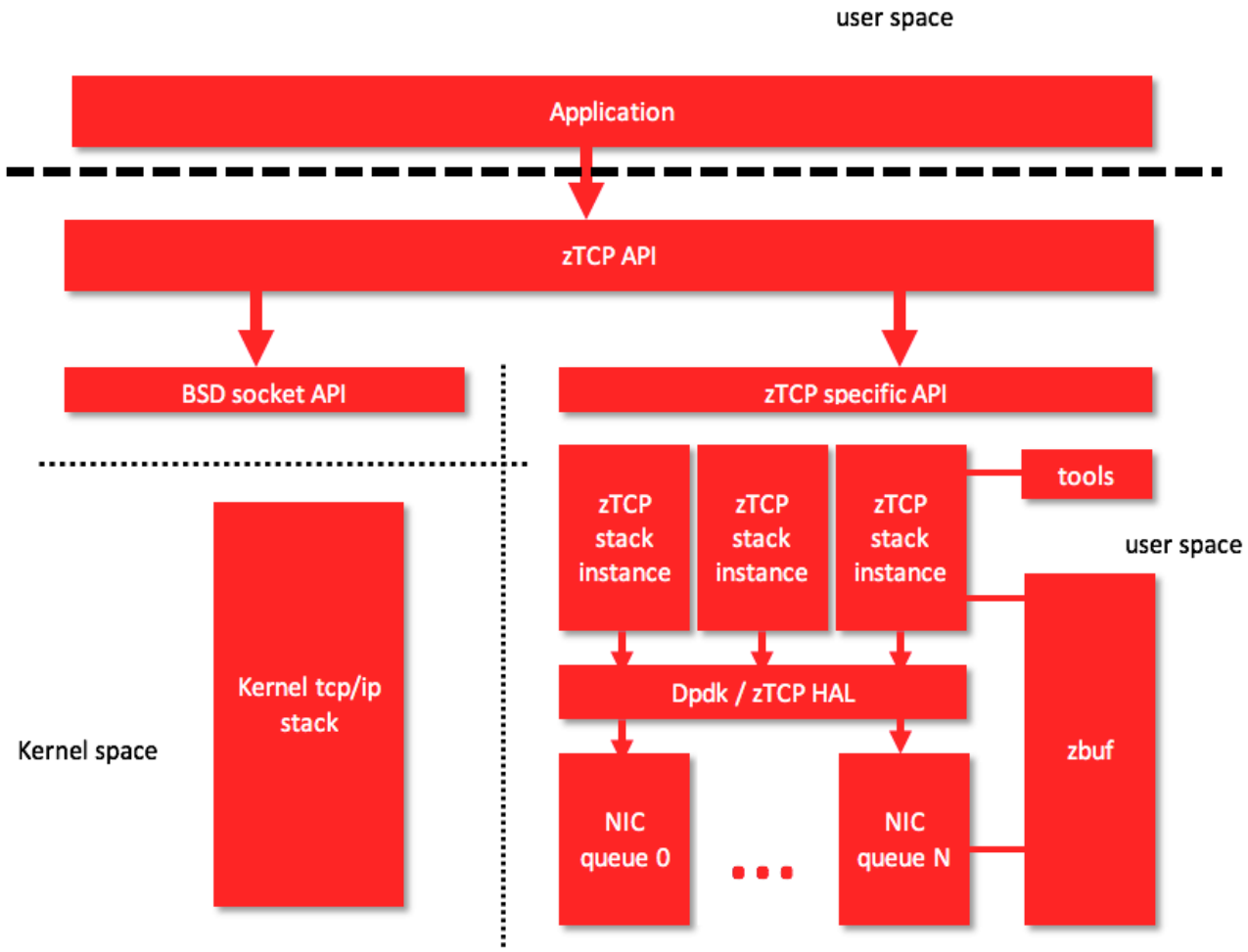
- CBS云盘技术架构
  - 采用SPDK，IO请求在用户态处理
  - CBS SDK负责IO业务层的处理
  - 最终由内核协议栈发送到存储集群

- 性能瓶颈
  1. 用户态/内核态切换开销
     - 中断影响
     - 内核模块的影响
     - 跨核处理
  2. 多核扩展性较差
  3. 大量CPU消耗在内存拷贝中



整体架构

- **multi-instance协议栈**
  - 协议栈支持创建多实例，即应用程序可以调用API创建多个stack，每个stack可以对应不同实现，应用使用在不同的stack上创建和使用socket，不同stack互不影响
- **报文处理无锁设计**
  - 每个stack实例仅在一个cpu上进行处理，结合dpdk使用，使用polling mode，每个处理逻辑独占一个cpu
- **协议栈全程零拷贝**
  - 发送数据时应用程序的buffer不需要copy到协议栈内；接收数据时，协议栈直接将网卡buffer给到应用程序
- **协议栈API封装**
  - API同时支持zTCP和BSD socket API，提供了对应用透明的封装
- **协议栈工具**
  - 支持各种协议栈工具，抓包，flow，统计等工具查看协议栈内部状态

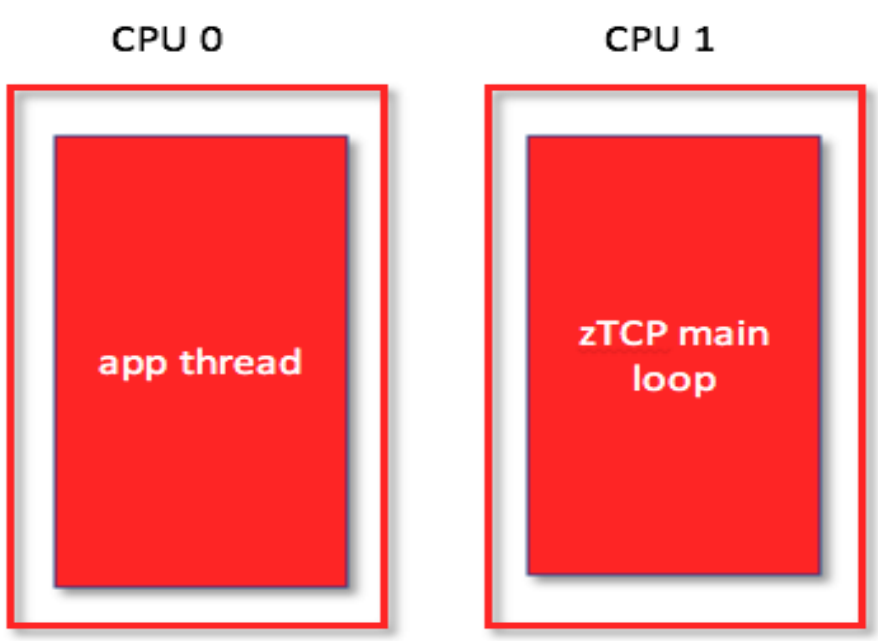


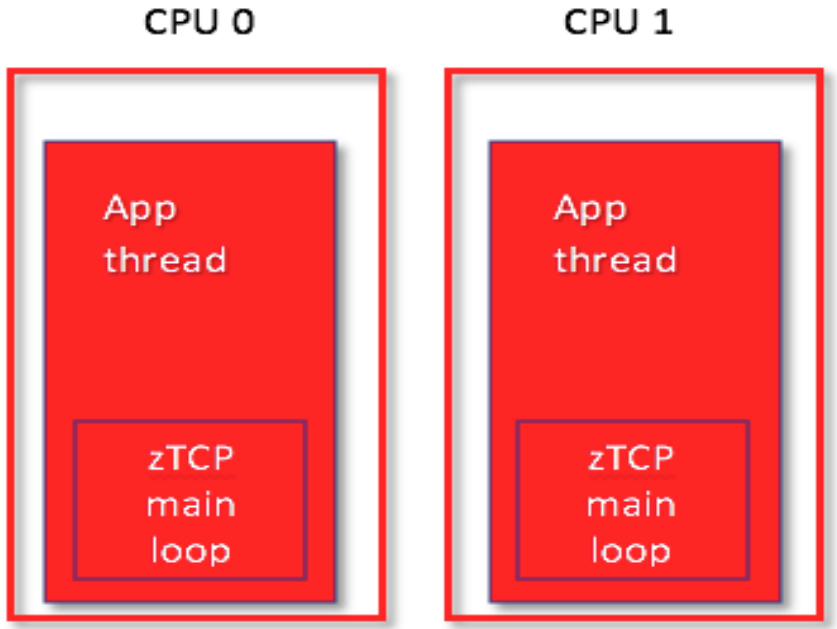
zTCP stack

- 主要协议特性：
  - 802.3 ethernet
  - IPv4，静态路由配备，动态ARP配置
  - TCP/ICMP
  - TCP主要特性：
    - timestamp
    - window scale
    - mss
    - 拥塞控制: reno
  - 使用dpdk pmd网卡驱动
  - 独占CPU运行
  - non-blocking socket
  - 对外查询接口

zTCP threading model

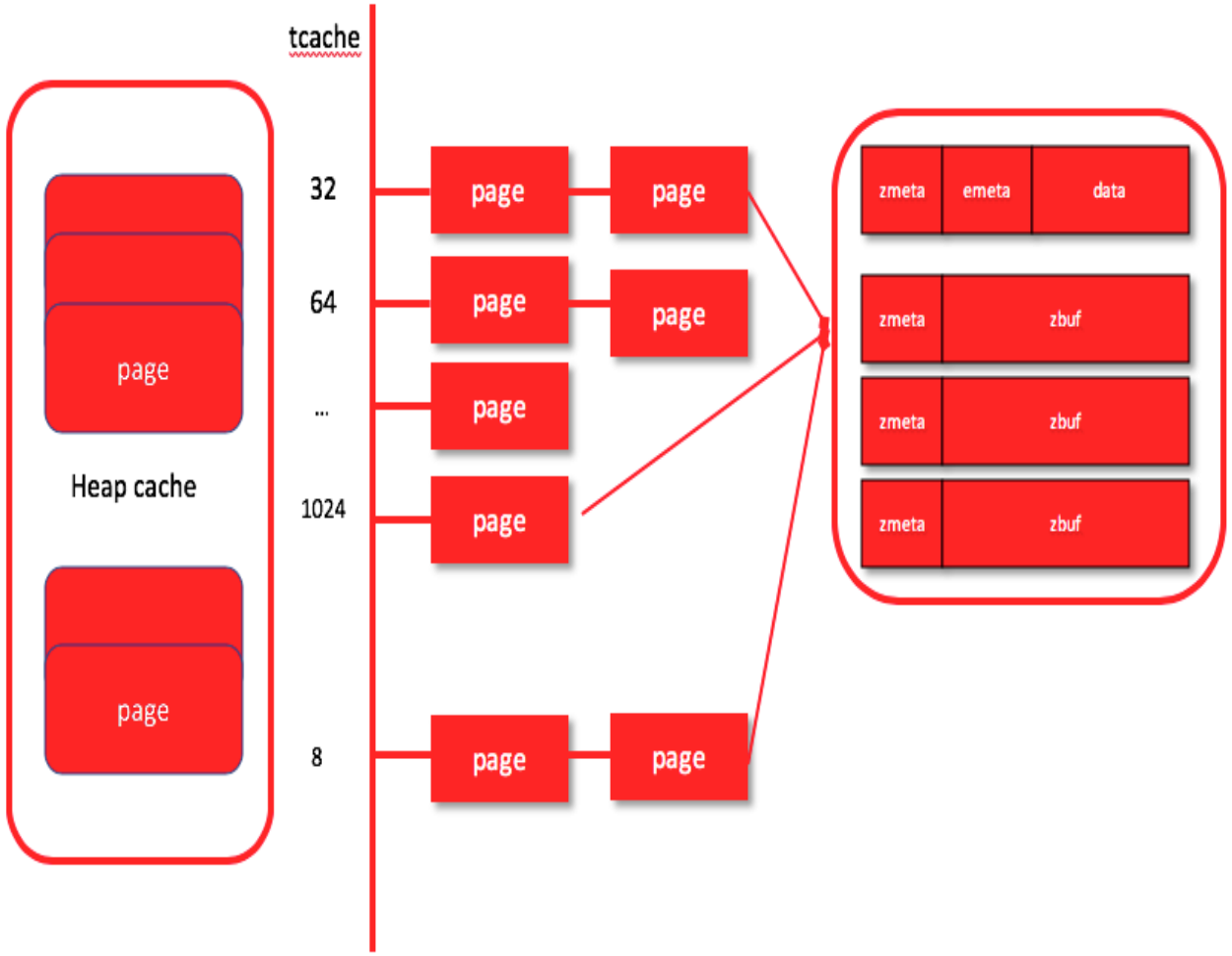
- 一期二期采用app thread和ztcp thread分离的模式，缺点是需要额外的线程间通讯，cache-miss较高，水平扩展能力较差
- 三期采用app thread和ztcp main loop结合的方式，协议栈可以跟随app进行扩展，线程通讯开销完全消除





zbuf

- 全新内存接口
  - zbuf_malloc/zbuf_free
  - zbuf_mem_pool
- 建立在大页之上，大页独立分配，建立在dpdk之外
- per-cpu local cache，尽量保证无锁分配（centrel cache是有所的)
- 支持协议栈零拷贝特有功能
  - buffer ref-counting
  - 从任意offset找到buffer metadata



零拷贝

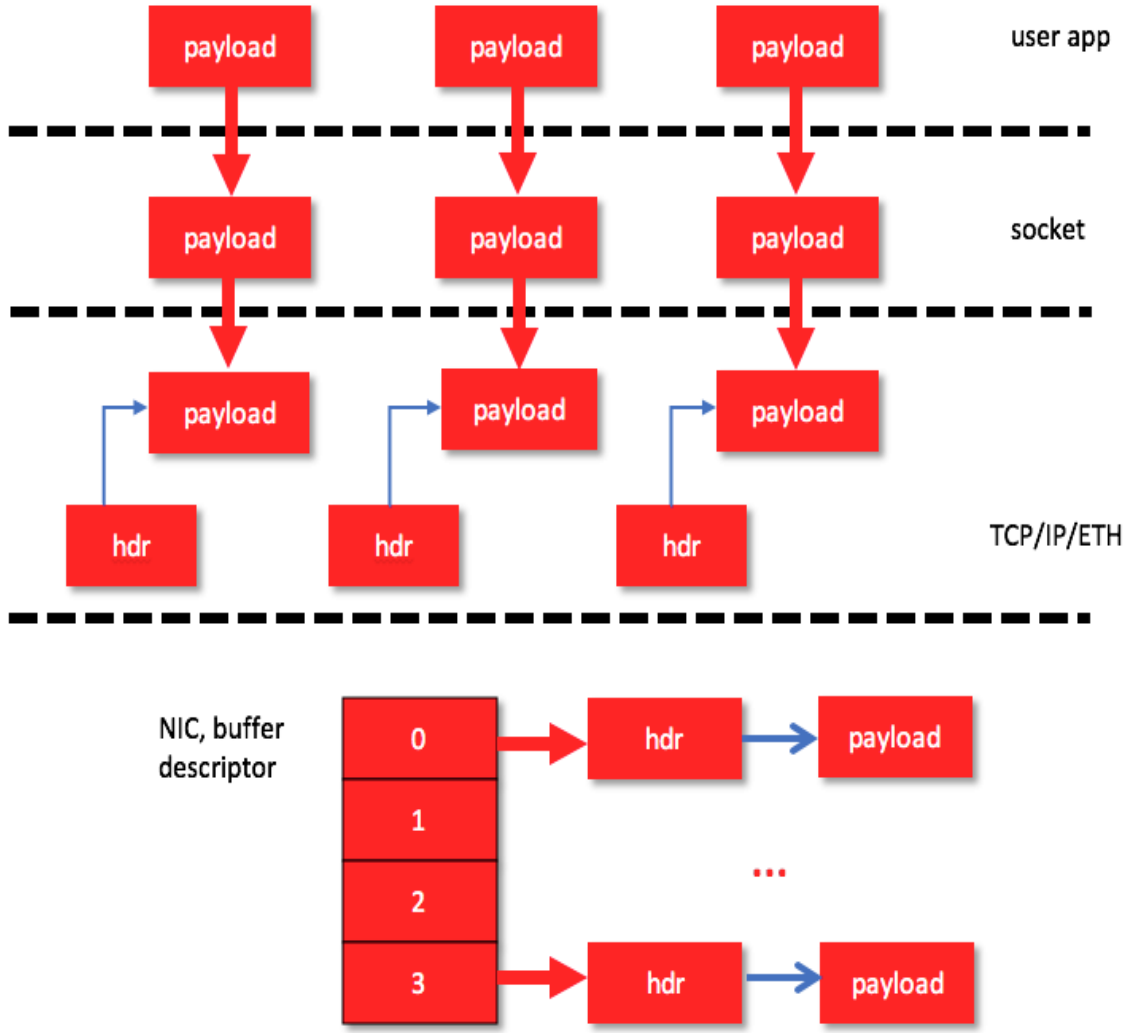
- 协议栈全程零拷贝
  - 应用程序发送数据时协议栈不进行拷贝
  - 应用程序接收数据时协议栈不进行拷贝
- 内核协议栈需2次拷贝
  - 发送时用户态buffer到内核skb需要进行拷贝
  - 接收时内核skb到用户态buffer需要进行拷贝
- Memory ownership – 内存属于谁?
  - 对于非零拷贝socket, write/send/read/recv返回即可
  - 内存一般来说谁分配谁释放
  - 对于零拷贝socket，需要重新定义
  - 发送涉及到tcp重传
  - 接收涉及到如何传递buffer

零拷贝- memory ownership

- 发送方向内存由应用使用zbuf分配
- 应用程序使用setsockopt来设置callback
- 当应用程序使用writev/write发送完成后，内存属于ztcp，应用不能再做任何操作
- 协议栈使用完成后调用应用程序callback来回收内存
- 接收方向内存由ztcp协议栈使用zbuf分配
- 应用使用readv/read接收数据后，内存属于应用程序，应用在合适的时间释放

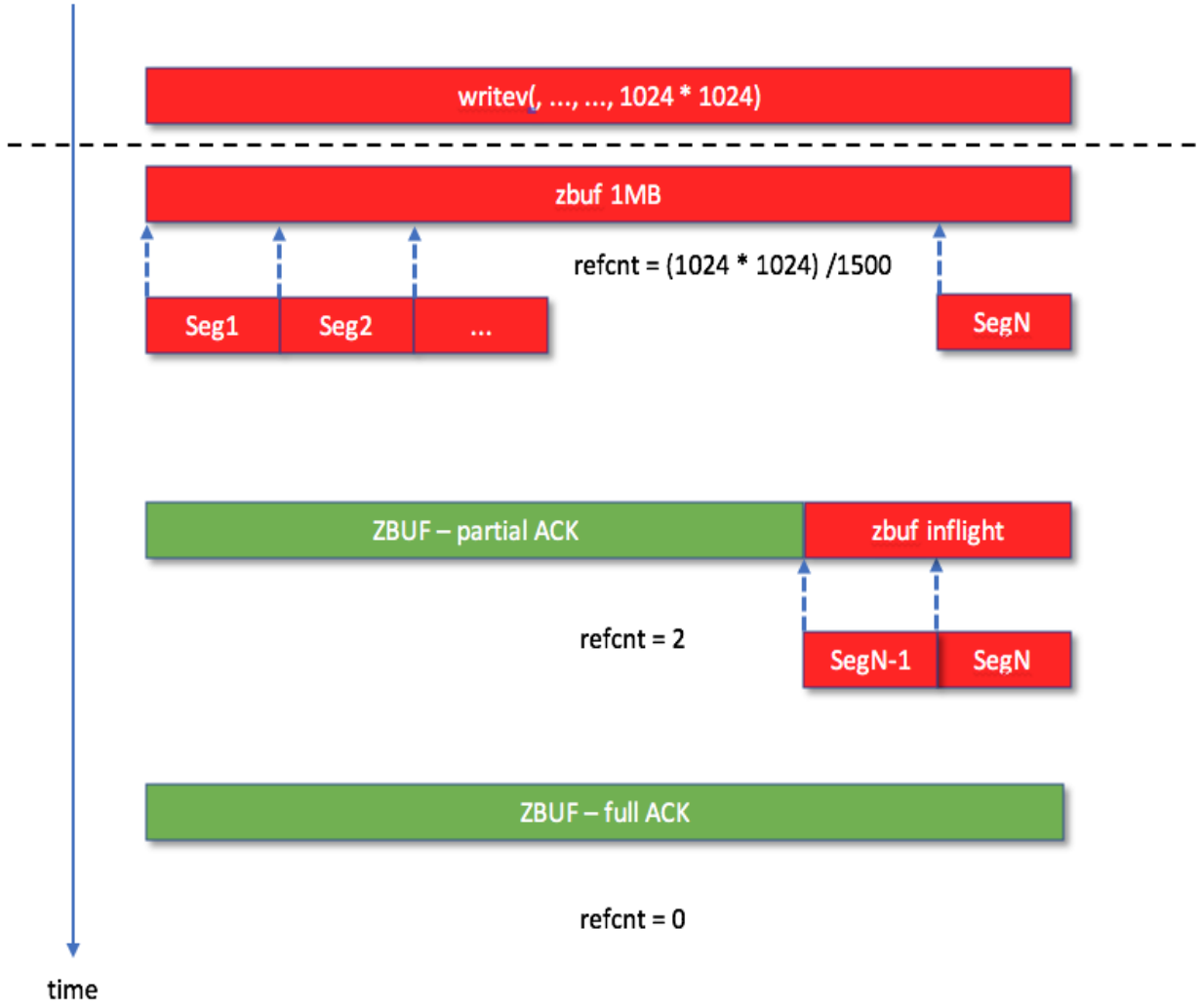
数据流 – 发送方向

- 应用程序发送数据时，ztcp直接将payload放入socket send buffer中，ztcp不做数据拷贝
- ztcp在发送数据时，分配并填写协议头部，之后将payload和协议头部串联起来形成一个完整报文，提交给网卡驱动
- ztcp收到ack后调用callback来释放payload应用程序读取数据后需要及时释放iov



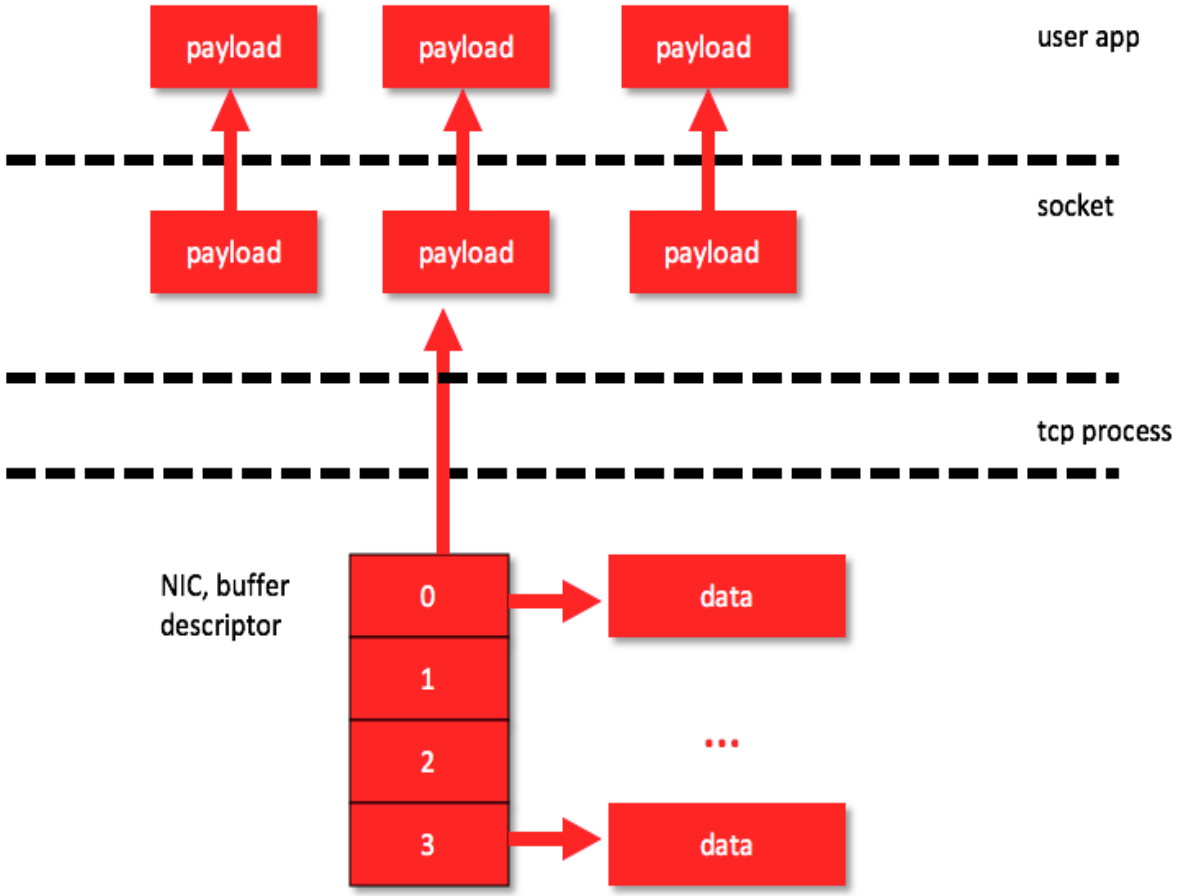
数据流 – 数据切割

- 协议栈一次能发送的数据大小受到多个因素的限制
  - MTU
  - 拥塞窗口
  - send buffer大小
- 当用户数据大小超过当前限制，就需要切割，分批次发送
- 对端回复ack有也可能分批确认，在整个buffer都被ack之前，应用还不能释放内存
- ztcp使用zbuf提供的refcount功能来确定何时释放通知应用来回收内存



数据流 – 接收方向

- ztcp从网卡驱动接收到报文后，首先经过协议处理，去掉协议头后，将payload放入socket层recv buffer中
- 应用程序通过调zTCP API接口收到一组iov
- 应用使用完成后调用zbuf接口及时释放内存

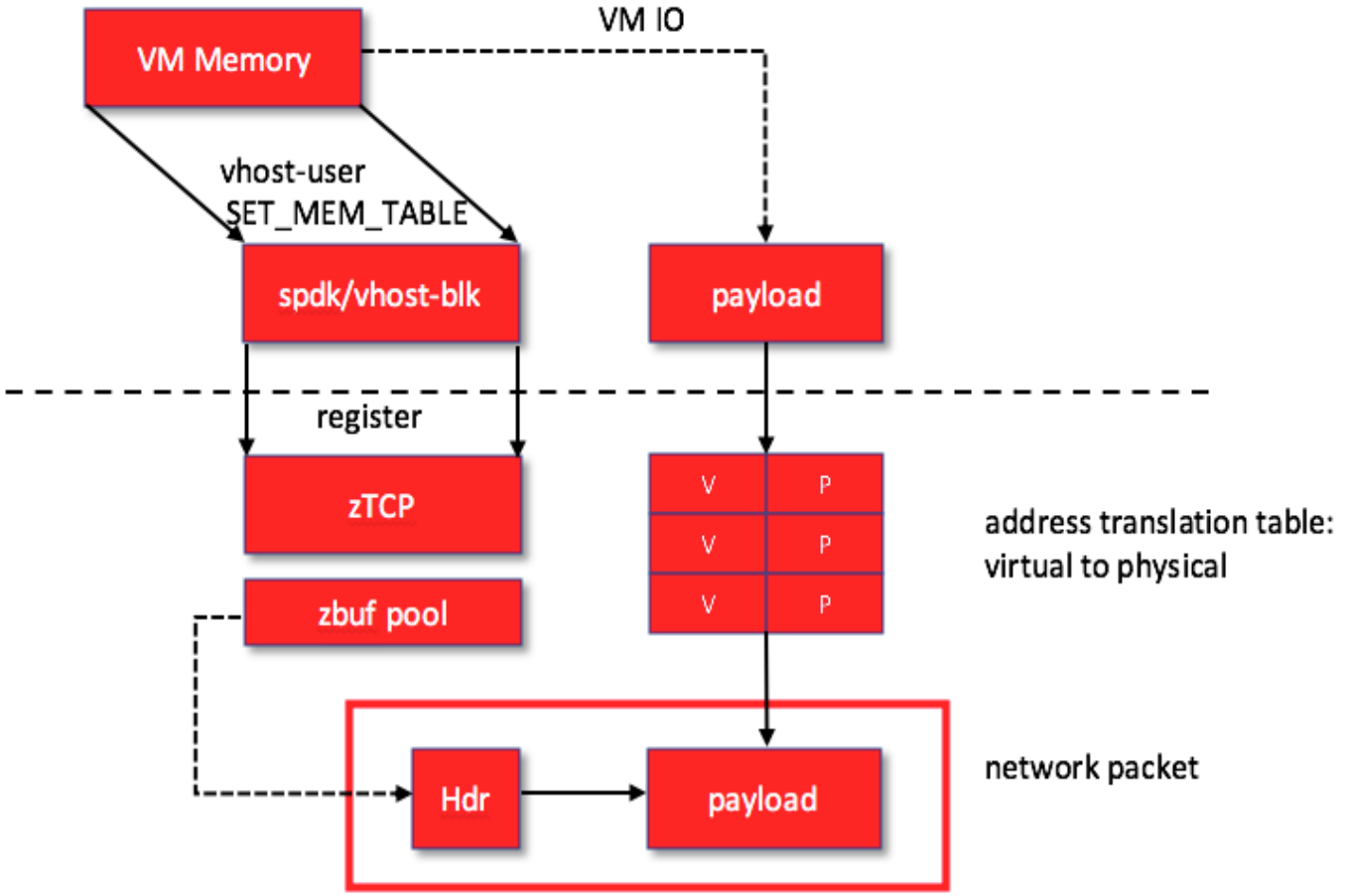


zTCP tools

- 抓包工具：ztcp内置tcpdump支持
- 链接工具：sstat可以查看tcp链接信息
- 协议栈统计工具：sstat/tcpstat
- 网卡统计工具：xstat/hw_stat
- 内存大页使用工具：heap_memestat
- ARP monitor工具：监控ARP变化，及时更新

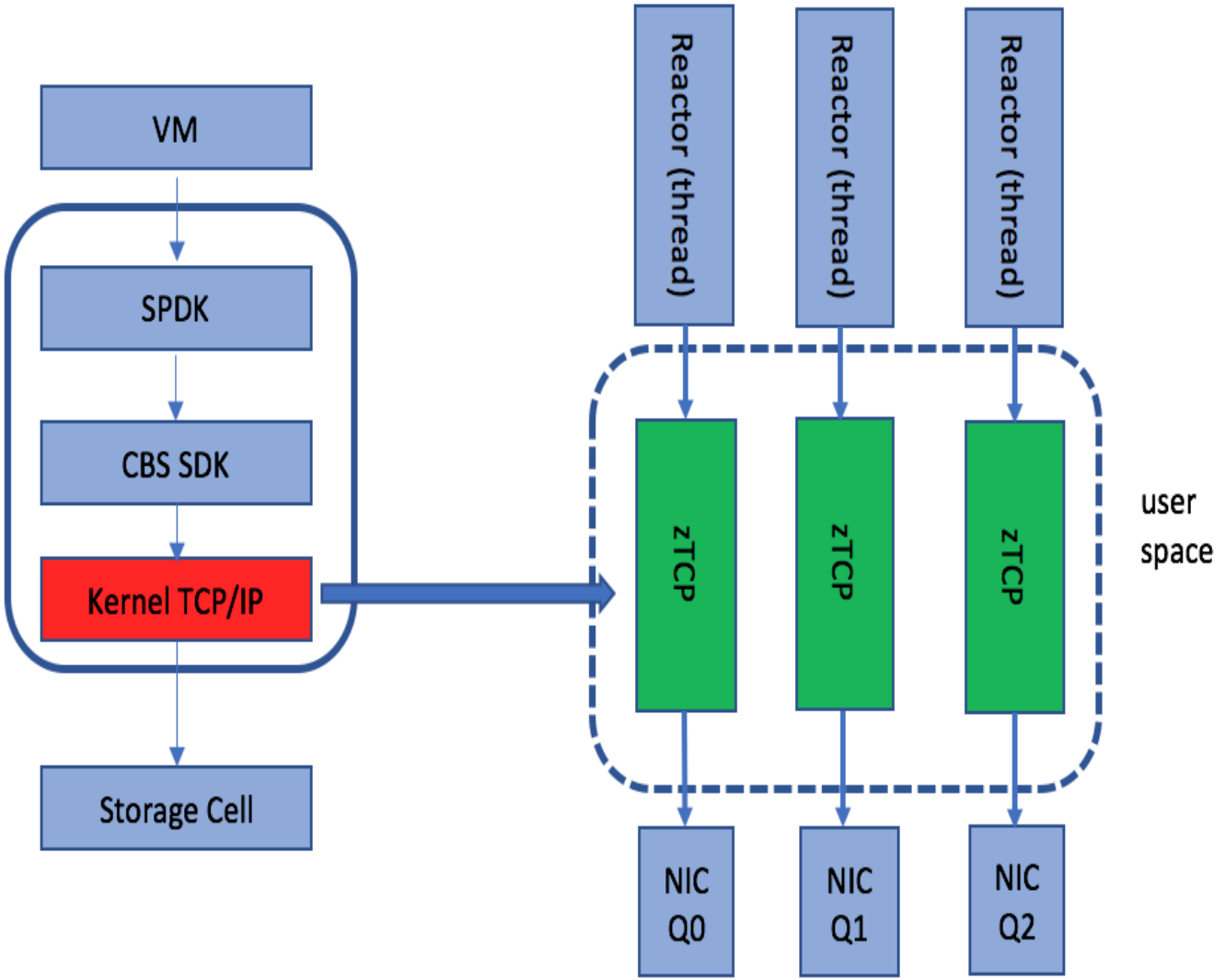
全路径零拷贝

- VM通过vhost-user协议大页内存注册到spdk中
- ztcp通过监听spdk处理逻辑，将子机内存注册到协议栈中
- ztcp内部保存子机内存虚拟地址到物理地址的转换表
- 子机发送IO请求，ztcp对子机内存区别处理
  - payload指向子机内存，不需要引用计数
  - hdr指向zbuf分配的内存，需要引用计数



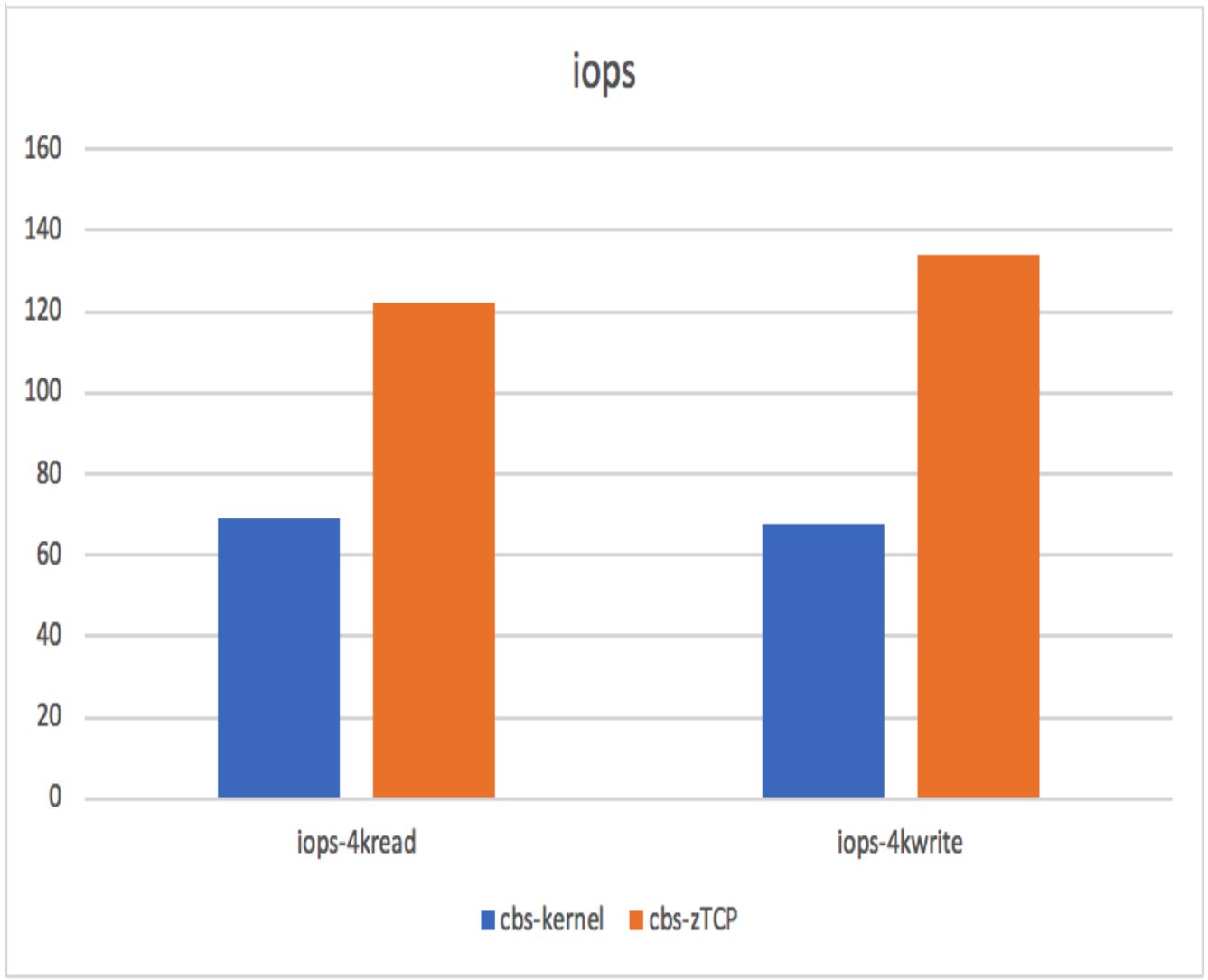
ztcp与spdk结合

- ztcp 提供stack loop API
- stack loop作为spdk reactor的一个poller定期运行
- poller初始化时由应用层创建
- 每个reactor上有一个ztcp poller



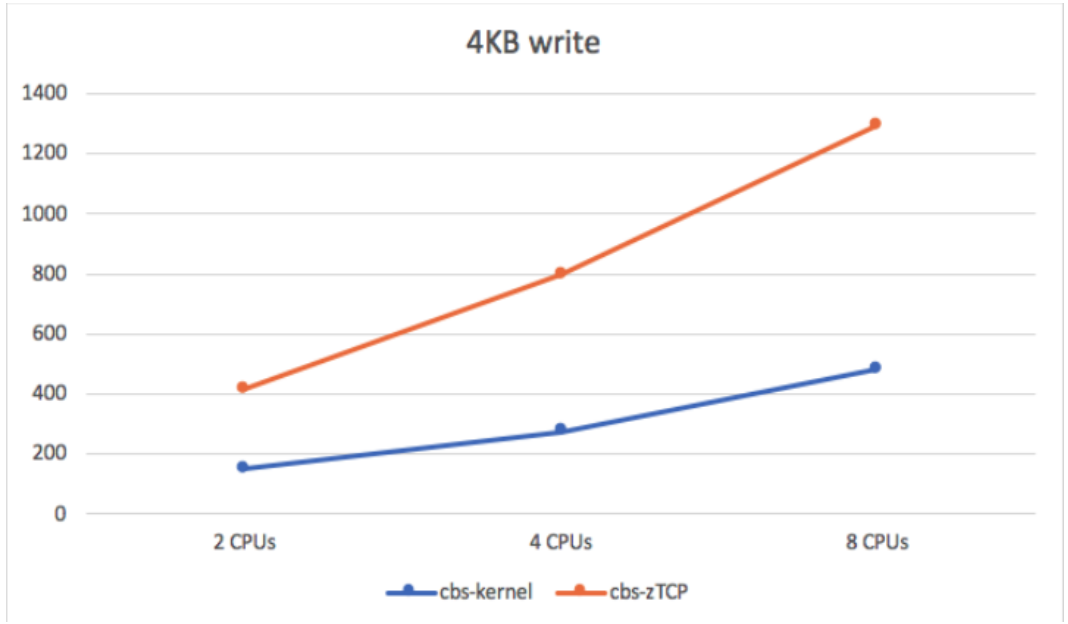
测试数据对比 – iops对比

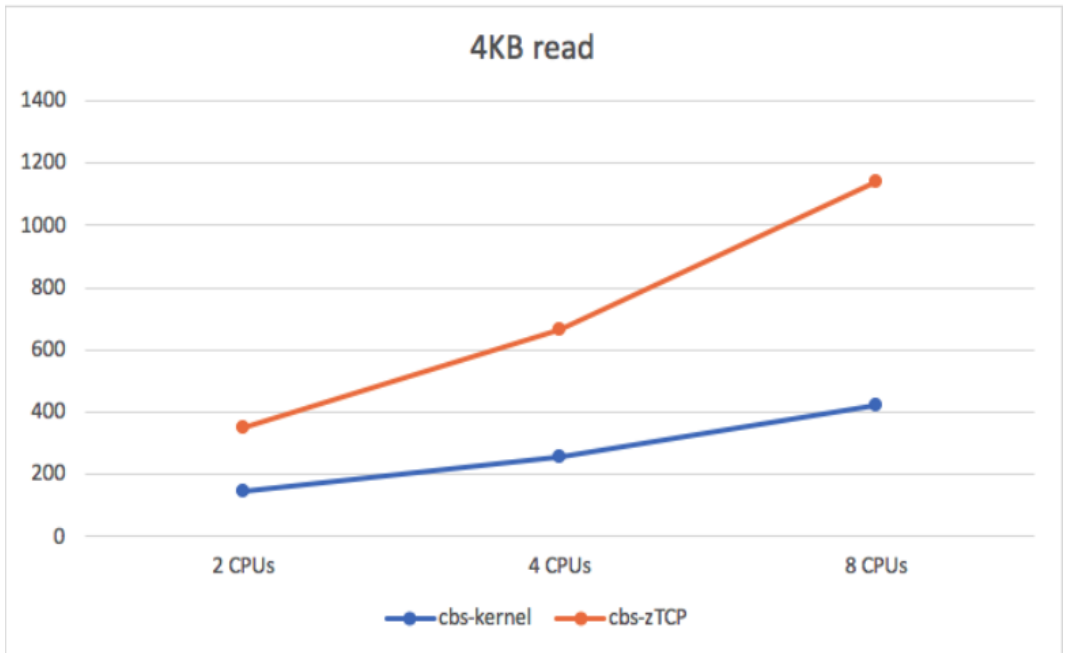
- fio 4KB随机读iops提升76%
- fio 4KB随机写iops提升97%



多核扩展测试

- 测试均在物理cpu上进行
- Kernel在8核下的性能和zTCP 4核性能相当
- 由于vhost rx方向无法做到完全零拷贝，所以测试结果读比写稍差





感谢各位