KRCore - Enhance RDMA Support in Kernel Space

1. 简介

传统RDMA library (ibverbs) 都是在用户态下直接进行数据传输，但是内核旁路 (kernel-bypass)同时也丧失了内核在资源共享方面的优势。传统RDMA程序中，多个线程之间的网卡资源不能共享，会给网卡带来较大的存储负担；同时每当需要基于RDMA进行通信时，每一个线程都需要建立一次握手连接(约18毫秒)，这对于生命周期较短的应用来说(例如FaaS)，建立连接的时间占比就非常显著了。对此我们希望在内核态支持RDMA通信，并通过设计来让网卡资源进行共享。这样一方面能够极大减少网卡的存储负担，另一方面也能够将握手的时间从十毫秒级别，降低到微妙(first handshake)甚至纳秒级别。

2. RDMA

RDMA (*Remote Direct Memory Access*)因其高吞吐、低延时的特性，被广泛应用于现代数据中心当中。相比于传统TCP/IP协议，RDMA不需要经过操作系统内核 (**kernel-bypass**)，能够在用户态进行本地、远端数据的直接访问。同时网卡（RNIC）能够绕过远端主机CPU (**cpu-bypass**) 直接访问远端主机的系统内存。

RDMA因不需要经过内核，也就避免了报文的kernel-user拷贝和系统调用的开销。同时数据传输过程中不需要CPU参与，可以提高CPU的使用效率。基于如上的种种优势，RDMA网络能够达到微秒（μs）级别的通信时延，网络峰值吞吐能够达到100Gbps甚至200Gbps。

2.1 名词介绍

RDMA通信协议可以分为如下三种：

- IB(InfiniBand): 链接层原生支持RDMA协议，能够天然支持远端内存直接访问。使用专有的IB网卡和IB交换机
- RoCE (RDMA over Converged Ethernet): 是为传统Ethernet提供的RDMA语义，交换机支持以太网传输，服务器使用相关的RoCE网卡。RoCEv2基于UDP协议。
- iWARP (Internet Wide Area RDMA protocol): 基于TCP/IP协议的RDMA通信技术，交换机支持以太网传输，仅网卡是特殊的支持iWARP网卡。

RDMA协议组成元素：

RDMA使用 *queue pairs*（QP） 来进行通信，每一个QP包含了一个 send queue (SQ) 和一个 receive queue (RQ)。RDMA协议中的数据访问操作包含了 READ (读)、WRITE(写)、SEND/RECV(消息)等。

其中读写操作均为cpu-bypass的单边(one-sided)操作，收发消息则是双边操作（接收端需要主机CPU参与）。RDMA QP类型包含了 RC (reliable connect), UC (unreliable connect), UD(unreliable datagram)，各个QP类型所支持的操作类型参见下图[1]：

	SEND/RECV	WRITE	READ
RC	✓	✓	✓
UC	✓	✓	✗
UD	✓	✗	✗

RC可以类比于传统TCP协议，UD则可以类比于传统UDP协议。在RC通信过程中，每创建一个本地RCQP都需要在远端网卡建立一个对端RCQP （类比单播）；而UD则没有这一个限制（类比多播）。

后文将会提到，RC这种“一对一“的通信模式会带来很大的性能问题

RDMA协议通过memory region (MR)来做内存访问的权限控制和地址翻译，用户程序可以将某一段虚拟地址空间以MR的粒度注册至网卡，此后用户程序就可以通过MR key为句柄来进行数据访问和数据交换。

2.1 Scalability问题

RNIC网卡中需要缓存QP的元数据信息、MR的元数据信息和MR缓存的PTE信息以加速网卡吞吐速度。当需要缓存的数据总量变大时，cache miss会引发RDMA性能的降低。换句话说，当连接数增多、网卡注册的MR数量增多，scalability就变成了RDMA性能的瓶颈。

3. KRCore

3.1 Proposals

资源复用

假设某一个场景中，m 台客户机与 n 台服务器进行通信，每一台客户机器上拥有 a 个用户程序，每一个用户程序平均使用 t 个线程。那么该场景中，一共需要 $n \times m \times a \times t$ 个RC QP来支持通信。我们希望能够对单个主机内的QP进行共享（单机跨进程、跨线程），将每一台主机内的RC QP总量设置为 K （常量）个，那么仅需要 $K \times n \times m$ 个 RC QP，只与集群规模有关。

从设计思路上，KRCore提出虚拟QP (Virtual-QP)的概念，将用户态的多个QP映射至内核态的一个真实物理QP，以实现QP资源的共享。

连接复用

传统用户态RDMA进行握手需要创建新的QP资源、初始化QP状态，这一握手过程需要十毫秒级别的时延。而当建连的并发量上升后，单个主机至多只能处理500~600个握手连接请求，这与RDMA处理数据传输请求的吞吐量是完全不平衡的。特别是在握手建连操作和数据传输操作数量接近的场景下，处理握手就会变成瓶颈。

KRCore引入了RCQP**共享连接池**的设计，客户端只需要知道server侧的元信息，就可以实现和远端的快速握手，这一过程的时延能够达到微秒级别。经过KRCore的优化，与远端握手需要的时延不超过10μs（相比于用户态的18ms提升了四个数量级），由于共享连接池的存在，后续其他用户程序/其他线程在尝试建立相同的连接时，可以直接复用连接池内已存在的连接，而不需要重复进行握手。

综上小结，KRCore实现了微秒级的首次握手连接时延和连接池的复用。

全面拥抱Rust

KRCore完全由Rust代码（10K LOC）编写，能够以kernel module的形式装载入linux `4.15.0-46-generic` 内核。Rust语言因其内存安全的特性，被越来越广泛地应用在OS开发当中，可以很好地缓解C/C++语言中的内存泄漏问题。

3.2 挑战

系统调用负载

传统TCP/IP协议中，系统调用所带来的模式切换具有一定的overhead，经过实验我们得到每一次系统调用会带来0.4 μs的时延，使得KRCore的性能较用户态传统RDMA性能会降低约20%~30%。

资源保护

将单个物理QP资源共享给多个应用程序/多个线程，势必需要考虑资源的隔离与保护问题。如果单个QP接受到了错误请求，该QP会被设置为error状态，需要重新进行创建、建连，这无疑是非常耗时的。KRCore给出了一种纯软件层面实现的QP保护策略来规避如上的问题。

3.3 接口说明

在KRCore以内核模块的形式载入kernel之后，用户态程序只需要通过系统调用（`ioctrl`）的方式来调用KRCore支持的服务。

- `qconnect`: 与远端某一个主机进行握手建连。如果该连接已在内核态存在，那么直接复用该连接
- `qpush`: 提交若干个请求给虚拟QP（VQ），语义与`ibv_post_send`相似
- `qpop`: 从VQ中pop得到一个completion queue element(CQE)，通常与`qpush`配合使用。若能够pop得到一个CQE，则表示`qpush`请求执行完成，语义与`ibv_poll_cq`相似
- `qpush_recv`: 可以类比`ibv_post_recv`，用以表明主机端可以接受新的消息
- `qpop_msgs`:与`qpop`相同，该函数也是获取一个CQE；略有不同的是，该函数从QP的receive cq中获取CQE，如果能够pop到一个CQE，表示主机端已经接收到了一个新的消息

Example:

```
int qd = queue();
qconnect(qd, ...);     /* Connect to another endpoint */

send_wr reqs[2];       /* Prepare request  */
qpush(qd, &reqs[0]);   /* Post request out  */
qpop(qd);              /* Pop result */
```

Reference

[1] Design Guidelines for High Performance RDMA Systems