USTEVENT RDMA

2021-3-17

# RDMA是什么

RDMA支持[零复制](https://zh.wikipedia.org/wiki/%E9%9B%B6%E5%A4%8D%E5%88%B6)网络传输，通过使[网络适配器](https://zh.wikipedia.org/wiki/%E7%BD%91%E5%8D%A1)直接在应用程序内存间传输数据，不再需要在应用程序内存与操作系统缓冲区之间复制数据。这种传输不需要[中央处理器](https://zh.wikipedia.org/wiki/%E4%B8%AD%E5%A4%AE%E5%A4%84%E7%90%86%E5%99%A8)、[CPU缓存](https://zh.wikipedia.org/wiki/CPU%E7%BC%93%E5%AD%98)或[上下文切换](https://zh.wikipedia.org/wiki/%E4%B8%8A%E4%B8%8B%E6%96%87%E4%BA%A4%E6%8F%9B)参与，并且传输可与其他系统操作并行。当应用程序执行RDMA读取或写入请求时，应用程序数据直接传输到网络，从而减少延迟并实现快速的消息传输。

但是，这种策略也表现出目标节点不会收到请求完成的通知（单向通信）等相关的若干问题。

# USTEVENT集成RDMA

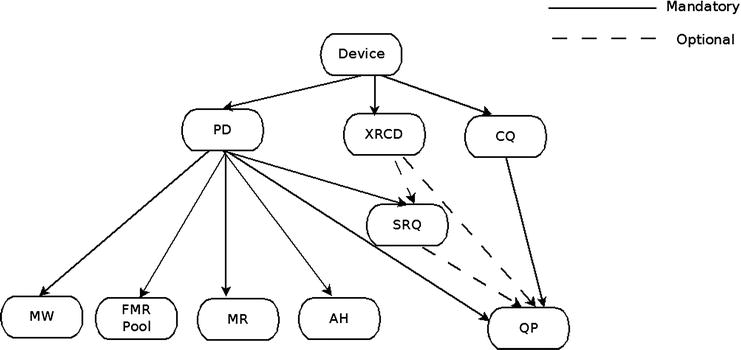
由于RDMA编程模型和传统套接字模型差异较大，细节较多，非常容易写出危机四伏的代码。因此将RDMA的连接管理，数据传输等功能集成到USTEVENT框架，封装充分验证过的高性能高可靠代码，将易用的接口暴露给开发者，减少重复劳动。目前所有实现只考虑支持RC服务类型。

USTEVENT对RDMA的支持主要分为以下几个功能：

1. RDMA运行时环境初始化，包括DEVICE初始化，PD初始化，GID选取，内存池创建，MR注册
2. QP管理，包括QP的创建、删除、连接建立、KEEP-ALIVE探测
3. 数据传输，包括发送数据、接受数据，目前只支持SEND

# RDMA初始化

## RDMA资源



RDMA初始化流程负责将DEVICE，PD，CQ，SRQ准备好，等到创建qp时再关联到对应的CQ和SRQ。此外框架还提供MR管理模块和内存池，允许应用直接使用已注册MR的内存池或自行分配内存自行注册MR。

## MR管理模块和内存池

条形图

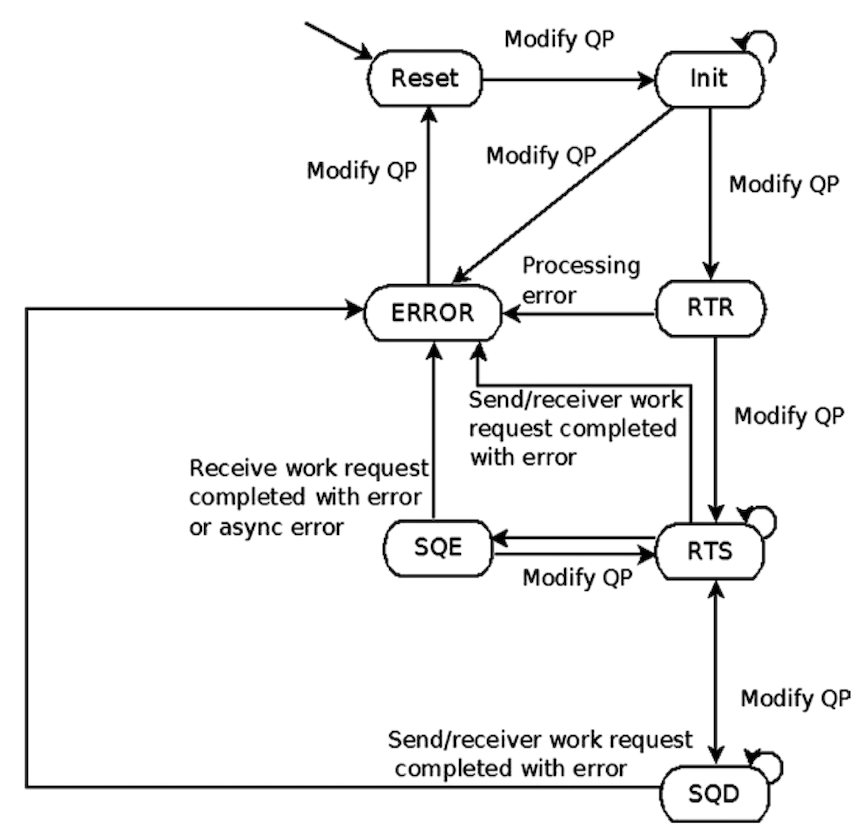
低可信度描述已自动生成

MR管理模块提供以下接口：

* 注册一段连续的2MB对齐的内存（注册MR并构造二级页表索引）
* 注销已注册过的内存（注销MR并删除二级页表索引）
* 通过已注册地址查找返回MR的lkey、rkey（二级页表索引）

其中，注册/注销接口提供给应用使用，应用将自行管理的内存注册后即可调用框架提供的RDMA数据传输接口，无需关注lkey、rkey等细节，由框架自动通过地址查找。

# QP连接管理



上图是一个完整的QP状态转换图，USTEVENT的实现方式仅涉及其中INIT、RTR、RTS、ERROR四类状态，QP资源创建时初始为INIT状态，连接建立过程将QP连续地从INIT转换到RTR再到RTS。当连接断开时直接将QP转换为ERROR状态，等到WR全部完成后释放QP。重连场景不会复用同一个QP，而是先断开连接释放一个QP，再重建一个QP。所以在USTEVENT框架中，QP的状态转换过程是一条单向链，INIT->(RTR)->(RTS)->(ERROR)。

## QP连接建立

图示, 示意图

描述已自动生成

QP连接建立过程最重要的过程是双方互相得到对方的关键三元组{lid，gid，qp\_num}，USTEVENT的实现是需要进行RDMA的双方通过一条TCP连接来交换三元组，当两方都通过ibv\_modify\_qp转换到RTS状态，连接就建立成功了。

TCP连接上除了交换三元组，可以增加更多可协商的信息，包括双方支持opcode，SEND最大支持大小，框架新增特性兼容等。有了这些信息，未来可以在代码中做到同时向前和向后兼容，减少不同版本的依赖。

## QP连接断开

图示, 示意图

描述已自动生成

主动断开连接第一步是将QP直接修改为ERROR状态，随后所有在途的传输都会返回失败，而对端只有发送消息失败才能感知到连接断开。

框架通过IBV\_EVENT\_QP\_LAST\_WQE\_REACHED感知连接断开，此事件表示对应的QP不会再消耗SRQ的recv WR。但此时不能立即销毁QP，虽然recv WR不会再关联到即将销毁的QP上，但可能存在仍未处理的send wr，若此时销毁QP，会造成资源的永久泄漏。

借助于RC保序的特性，框架通过ibv\_post\_send一个wr\_id为特殊标识的WR，此WR的完成意味着所有前序WR的完成。当框架从CQ中poll到这个特殊的wr\_id，说明所有send WR已经处理完毕，可以销毁QP。

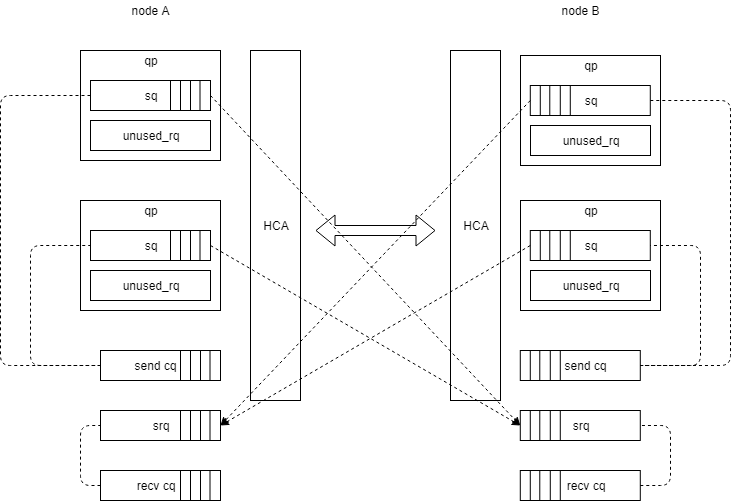
## KEEP-ALIVE

QP连接可能因为主动断开、进程退出或机器宕机等原因断开，在没有outstanding WR的时候，本地是无法感知对端QP的状态的。因此，可能这种情况，对端已经断开连接很久了，但因为后续这一对QP上一直没有传输数据，永远无法感知到连接断开，造成资源的泄漏；或过了一段时间，突然发送了大量的数据，但因对端QP已经关闭，所有WR都报错失败了，数据传输过程必须等待新的QP重新建立，造成不必要的延迟。

KEEP-ALIVE机制通过zero byte messages定期地探测QP的连通性，具体做法是框架n次轮询后，对每个存活的QP下发不带数据，wr\_id为特殊值的RDMA WRITE。若从CQ中得知该wr失败，则认为连接已断开，进入连接断开处理流程。其中RDMA WRITE是单边操作，因此不会消耗对端的recv WR。

# RDMA传输

## 消息语义SEND/RECEIVE



SEND/RECEIVE操作是一一对应的，发送方的每个send WQE在接受方都会产生一个recv CQE，由于网卡收包是异步的，所以recv WQE必须提前加入recv queue提供给网卡，因此随着QP数量的增加，recv buffer可能会消耗并浪费大量内存资源。为了解决该问题，框架使用SRQ负责接收所有QP的数据，所有QP在创建时都指定相同的SRQ。

框架每次轮询处理完一个recv CQE，都会尝试post一个recv WR到SRQ中，以确保收包不会因WQE不足而失败。

因为SEND的成功发送依赖于对端的recv WR的buffer至少等于send数据大小，而recv端采用SRQ，无法确定下一个recv WR被用于哪一个QP，因此框架实现之初约定了最大的包大小为256KB，所有的recv buffer都至少大于256KB。另外，由于recv buffer对应的QP无法确定，导致了发送和接收的应用逻辑不对称：发送时可以使用应用自己申请的内存，也可以使用框架的内存池，确保零拷贝；而接收时，数据总是写入到框架的内存池中，若应用需要将数据写入自己申请的内存中，则必须经历一次拷贝。

## 内存语义READ/WRITE

RDMA READ/WRITE提供了读取/写入远端内存的能力，使用这种远端内存访问能力有前提条件：

1. 被访问端的Remote Memory已注册到QP所属的PD，并赋予进行Remote Read/Write的权限
2. 访问端需要知道Remote Memory的地址和具备相应权限的rkey(Remote Key)

框架支持READ/WRITE的可以带来的改变：

1. 数据接收方通过READ接收数据，降低recv buffer的内存资源
2. 针对某些应用场景，可以直接将本地buffer数据WRITE到远端目的buffer，或将远端buffer数据 READ到本地buffer，减少一次从recv buffer的拷贝（真正的实现零拷贝传输）

框架最早期曾经实现了一套RDMA READ方案，当payload size大于recv buffer size后，会通过RDMA READ进行收包。该方案可以减少recv buffer的内存资源，但仍然无法实现零拷贝，原因是应用无法指定收包的目的地址。另外，通过READ替代RECEIVE实现透明收包需要额外增加网络交互，READ是单边操作，目的端无法感知数据传输的完成，READ完毕后接收端需要发送一个ACK消息告知发送方数据传输完成，额外占用了带宽，并增加了时延。同理，如果使用WRITE来实现大包传输，目的端需要一个额外的ACK来感知收包完成。

针对已有ACK语义（每个请求都有响应确认）的应用来说，在请求协议中增加一些字段即可适配READ/WRITE语义。因此，框架可以提供轻量级的READ/WRITE支持，将READ/WRITE接口直接暴露给应用，让应用自行灵活适配，发挥最优的性能。

UDisk协议适配RDMA READ/WRITE可以参考Nvme over Fabrics的设计，大致流程如下图：

图示

描述已自动生成

图示, 示意图

描述已自动生成

# 参考资料

1. <https://zh.wikipedia.org/wiki/%E8%BF%9C%E7%A8%8B%E7%9B%B4%E6%8E%A5%E5%86%85%E5%AD%98%E8%AE%BF%E9%97%AE>
2. <https://link.springer.com/chapter/10.1007/978-1-4302-6197-1_13>
3. <https://insujang.github.io/2020-02-09/introduction-to-programming-infiniband/>
4. <https://www.rdmamojo.com/2013/09/20/zero-byte-messages/>
5. https://www.rdmamojo.com/2012/12/28/ibv\_destroy\_qp/