QCon+ 案例研习社

RDMA 的 Rust 安全实践

北京达坦科技 施继成

目录

1 RDMA 简介

2 Rust 语言简介

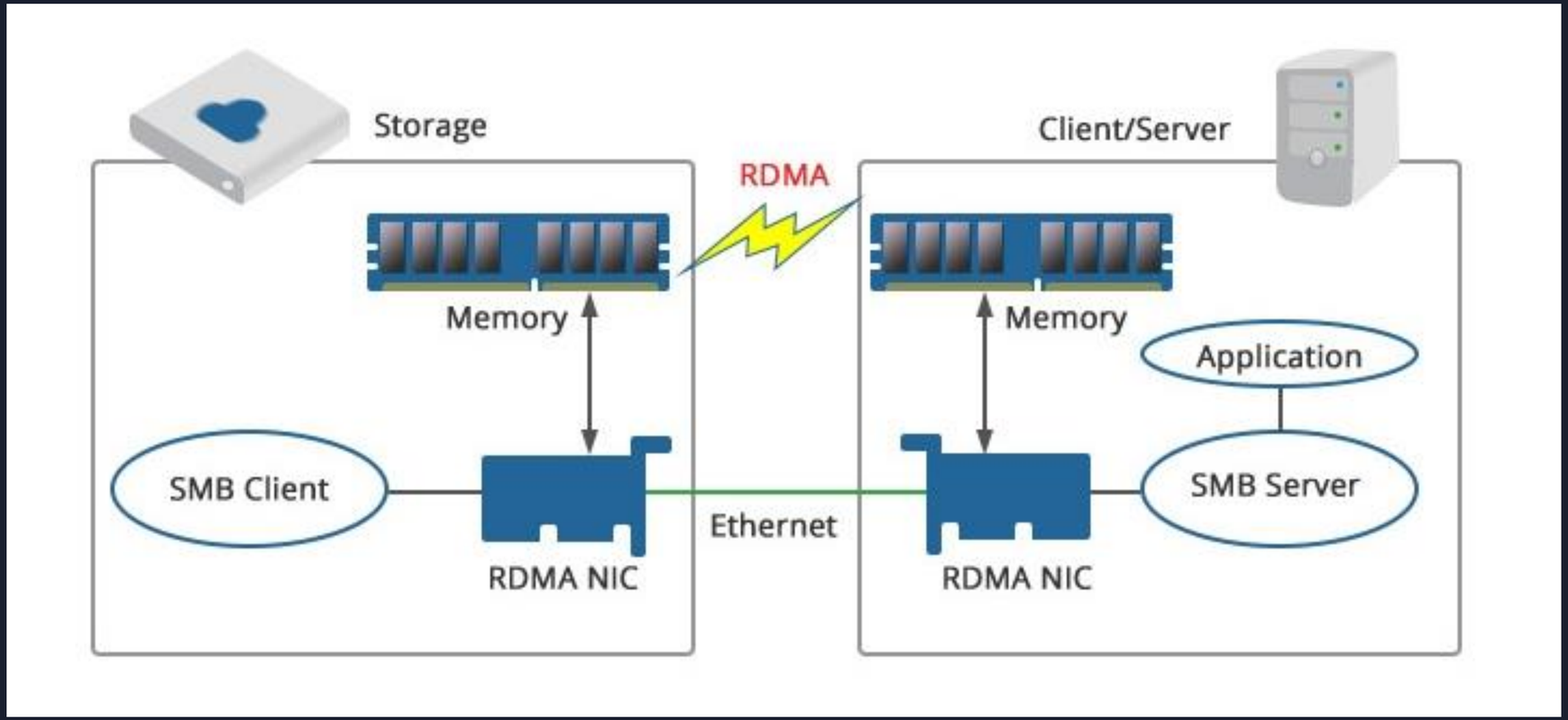
3 RDMA 的 Rust 封装设计

4 RDMA Rust 接口

RDMA 简介

天上掉下来的馅饼很美好

- 绕过系统内核
- 硬件处理，绕过 CPU
- 高速低延迟



RDMA 简介

天下没有免费的午餐

- 接口和现有网络接口不兼容
- 接口比较繁琐使用不易
- 出错时问题极难定位

RDMA 简介

Show me the code

- 一个简单的步骤需要数十行代码
- 设置参数繁复，报错信息极少
- 指针满天飞

```
101 static int pp_connect_ctx(struct pingpong_context *ctx, int port, int my_psn,
102                           enum ibv_mtu mtu, int sl,
103                           struct pingpong_dest *dest, int sgid_idx)
104 {
105         struct ibv_qp_attr attr = {
106                 .qp_state               = IBV_QPS_RTR,
107                 .path_mtu               = mtu,
108                 .dest_qp_num            = dest->qpn,
109                 .rq_psn                 = dest->psn,
110                 .max_dest_rd_atomic     = 1,
111                 .min_rnr_timer          = 12,
112                 .ah_attr                = {
113                         .is_global      = 0,
114                         .dlid           = dest->lid,
115                         .sl             = sl,
116                         .src_path_bits  = 0,
117                         .port_num       = port
118                 }
119         };
120
121         if (dest->gid.global.interface_id) {
122                 attr.ah_attr.is_global = 1;
123                 attr.ah_attr.grh.hop_limit = 1;
124                 attr.ah_attr.grh.dgid = dest->gid;
125                 attr.ah_attr.grh.sgid_index = sgid_idx;
126         }
127         if (ibv_modify_qp(ctx->qp, &attr,
128                           IBV_QP_STATE              |
129                           IBV_QP_AV                 |
130                           IBV_QP_PATH_MTU           |
131                           IBV_QP_DEST_QPN           |
132                           IBV_QP_RQ_PSN             |
133                           IBV_QP_MAX_DEST_RD_ATOMIC |
134                           IBV_QP_MIN_RNR_TIMER)) {
135                 fprintf(stderr, "Failed to modify QP to RTR\n");
136                 return 1;
137         }
138
139         attr.qp_state       = IBV_QPS_RTS;
140         attr.timeout        = 14;
141         attr.retry_cnt      = 7;
142         attr.rnr_retry      = 7;
143         attr.sq_psn         = my_psn;
144         attr.max_rd_atomic  = 1;
145         if (ibv_modify_qp(ctx->qp, &attr,
146                           IBV_QP_STATE              |
147                           IBV_QP_TIMEOUT            |
148                           IBV_QP_RETRY_CNT          |
149                           IBV_QP_RNR_RETRY          |
150                           IBV_QP_SQ_PSN             |
151                           IBV_QP_MAX_QP_RD_ATOMIC)) {
152                 fprintf(stderr, "Failed to modify QP to RTS\n");
153                 return 1;
154         }
155
156         return 0;
157 }
```

RDMA 简介

梦想中的满汉全席

- 接口简单易用
- 防止低级错误，即使出错报错明确
- 内存安全 + 线程安全
- 和 C 语言实现性能接近

Maybe Rust?

目录

1 RDMA 简介

2 Rust 语言简介

3 RDMA 的 Rust 封装设计

4 RDMA Rust 接口

Rust 语言简介

安全而高效的语言

- Ownership and Borrowing → 内存安全 + 线程安全
- Async / Await and Futures → 语言级别的异步编程接口
- 零成本抽象 → 更高的抽象级别，性能几乎没有损失

目录

1 RDMA 简介

2 Rust 语言简介

3 RDMA 的 Rust 封装设计

4 RDMA Rust 接口

RDMA 的 Rust 封装设计

从 C 到 Rust，Rust Binding

- 找合适的切口进行 C 到 Rust 语言的转换
  - 用 Rust 重新写 RDMA Ibvlib 工作量巨大
  - 将 Ibvlib 提供的接口和数据结构进行封装
- Rust bindgen 自动生成接口转换代码，但不是万能的
  - Inline function
  - Nested data structure
- https://github.com/datenlord/rdma-sys

Rust

```rust
#[inline]
pub unsafe fn ibv_wr_send(qp: *mut ibv_qp_ex)
```

C

```c
void ibv_wr_send(struct ibv_qp_ex *qp);
```

RDMA 的 Rust 封装设计

问题找到了，解决问题也不远了

➢ Ibv 接口的缺陷

- 裸指针
- 配置项过多，容易出错
- 内存使用不安全（网卡在用的内存程序是否能用？）
- 没有异步接口
- 本地内存管理效率
- 远端资源无法管理

RDMA 的 Rust 封装设计

化繁为简

- 用户初次使用往往不关注细节
  - 极简的默认配置
- 深度用户可以深度定制（逐步完善）
  - 为权限控制预留设置接口
  - 为性能关键参数预留接口
  - ……

RDMA 的 Rust 封装设计

所有权转移 —— 解决内存安全问题

- 基本原则
  - NIC 设备在使用的 RDMA 内存不能另做他用
  - 多线程、多任务间可共享 RDMA 内存块，但是需要被锁保护
- 接口设计
  - 每一个内存块被一把读写锁保护
  - 被硬件使用的内存块由后台任务持有（读或者写）锁

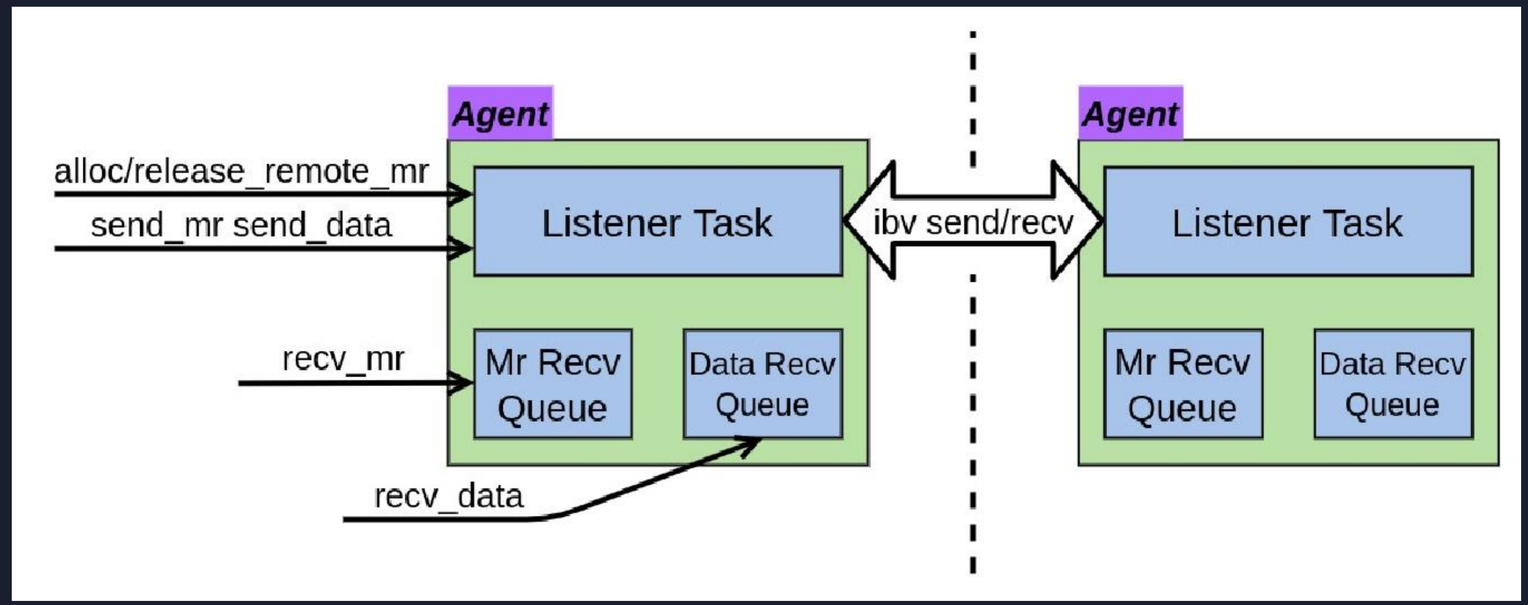
RDMA 的 Rust 封装设计

异步 I/O

➢ 使用异步的原因

- I/O 操作在异步环境下吞吐更好，CPU 开销更小
- Rust 语言有完善的异步支持，有成熟的异步 Runtime（Tokio）

➢ 异步实现架构



RDMA 的 Rust 封装设计

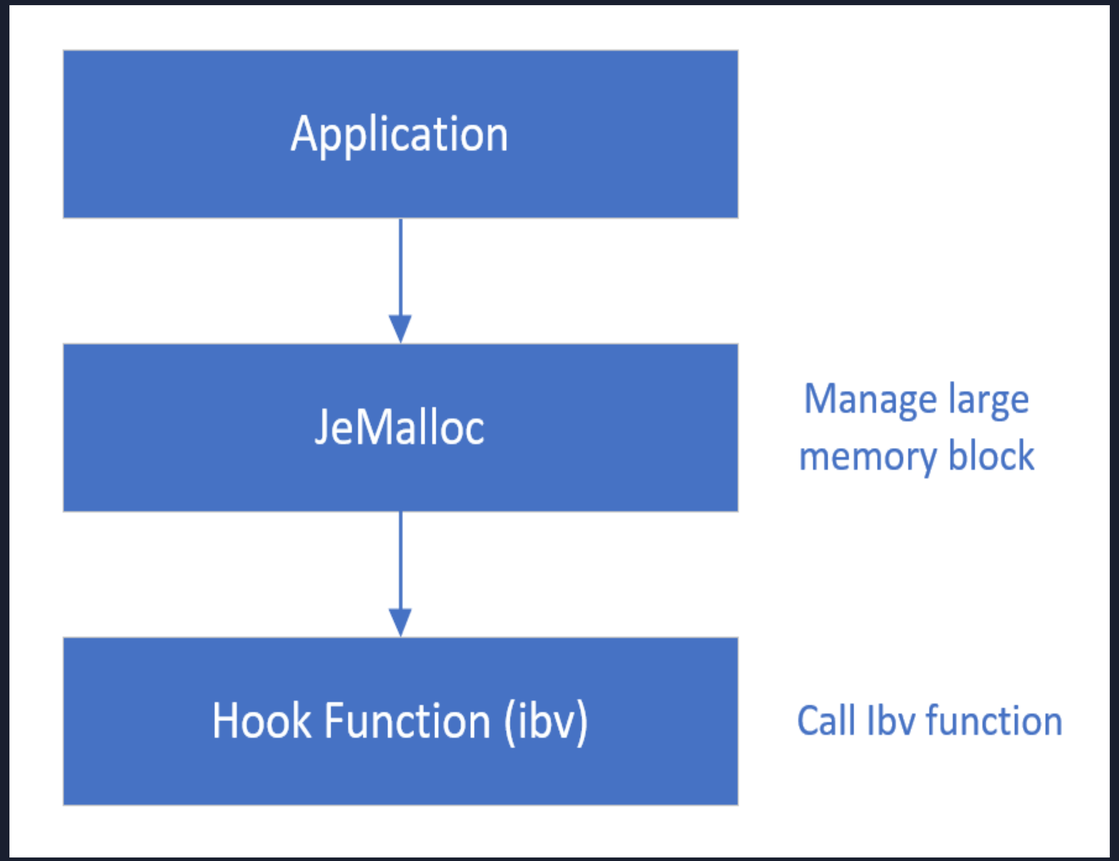
本地内存资源管理

➢ 为什么要管理本地内存资源?

- Memory Region 的注册和注销成本太大
- 一般申请一大块内存，用户自己负责管理和释放

➢ JeMalloc

- 成熟的内存管理器，具有较好的拓展性



RDMA 的 Rust 封装设计

远端内存资源管理

➢ RDMA 的操作需要双方进行内存准备

- RDMA Write 需要远端提供足够大的写入空间
- RDMA Read 需要远端将数据提前准备好

➢ 远端资源释放问题

- 为远端操作准备的内存何时可以释放?
- 显式地释放资源
- Timeout 保护，防止内存泄露

目录

1 RDMA 简介

2 Rust 语言简介

3 RDMA 的 Rust 封装设计

4 RDMA Rust 接口

Async RDMA 样例

- 建立连接

```
let rdma = Rdma::connect("127.0.0.1:5555", 1, 1, 512).await.unwrap();
```

- 申请内存资源

```
let mut rmr = rdma.request_remote_mr(Layout::new::<char>()).await.unwrap();
let mut lmr = rdma.alloc_local_mr(Layout::new::<char>()).unwrap();
```

- 发送数据

```
rdma.write(&lmr, &mut rmr).await.unwrap();
```

总结

- RDMA 接口复杂，需要安全的高级封装
- github.com/datenlord/async-rdma
  - 安全高效的内存管理
  - 简单高级的用户接口
  - 易用的异步接口

当造轮子无法避免时，努力造得好一点。

THANKS

QCon+ 案例研习社