o2iblnd -> verbs Overview

Overview

The o2iblnd module manages reliable connections using the verbs API. It sends and receives RDMA. It is used to interface with any HW interface which supports verbs, such as MLX, RoCE and OPA.

Network Startup

When an LNet network is initially added a device is specified using a device name, ex: ib0. The device is looked up using that name and the failover status of the device is determined.

```
struct net_device *netdev
netdev = dev_get_by_name(&init_net, ifname);
if (netdev == NULL) {
        dev->ibd_can_failover = 0;
} else {
        dev->ibd_can_failover = !!(netdev->flags & IFF_MASTER);
        dev_put(netdev);
}
```

Create and rdma ID

```
# define kiblnd_rdma_create_id(cb, dev, ps, qpt) rdma_create_id(current->nsproxy->net_ns, \

cb, dev, ps, qpt)
cmid = kiblnd_rdma_create_id(kiblnd_cm_callback, dev, RDMA_PS_TCP, IB_QPT_RC);
```

Bind to the IP address. We use IB over IP for connection establishment

```
rc = rdma_bind_addr(cmid, (struct sockaddr *)&addr);
if (rc != 0 || cmid->device == NULL) {
        CERROR("Failed to bind %s:%pI4h to device(%p): %d\n",
                        dev->ibd_ifname, &dev->ibd_ifip,
                        cmid->device, rc);
        rdma_destroy_id(cmid);
        goto out;
}
```

Protection domain creation

```
#ifdef HAVE_IB_ALLOC_PD_2ARGS
        pd = ib_alloc_pd(cmid->device, 0);
#else
        pd = ib_alloc_pd(cmid->device);
#endif
```

Listen for RDMA connections

```
rc = rdma_listen(cmid, 0);
```

Memory Registration

FMR or FastReg memory pools are allocated on startup

For FMR pool allocation:

```
struct ib_fmr_pool_param param = {
        .max_pages_per_fmr = LNET_MAX_IOV,
        .page_shift        = PAGE_SHIFT,
        .access            = (IB_ACCESS_LOCAL_WRITE |
                                        IB_ACCESS_REMOTE_WRITE),
        .pool_size»·····   = fps->fps_pool_size,
        .dirty_watermark   = fps->fps_flush_trigger,
        .flush_function    = NULL,
        .flush_arg         = NULL,
        .cache             = !!fps->fps_cache };

fpo->fmr.fpo_fmr_pool = ib_create_fmr_pool(fpo->fpo_hdev->ibh_pd, &param);
```

For FastReg allocation:

```
1568 #ifndef HAVE_IB_MAP_MR_SG
1569 »·······»·······frd->frd_frpl = ib_alloc_fast_reg_page_list(fpo->fpo_hdev->ibh_ibdev,
1570 »·······»·······»·······»·······»·······»·······    LNET_MAX_IOV);
1571 »·······»·······if (IS_ERR(frd->frd_frpl)) {
1572 »·······»·······»·······rc = PTR_ERR(frd->frd_frpl);
1573 »·······»·······»·······CERROR("Failed to allocate ib_fast_reg_page_list: %d\n",
1574 »·······»·······»·······»·······rc);
1575 »·······»·······»·······frd->frd_frpl = NULL;
1576 »·······»·······»·······goto out_middle;
1577 »·······»·······}
1578 #endif
1579
1580 #ifdef HAVE_IB_ALLOC_FAST_REG_MR
1581 »·······»·······frd->frd_mr = ib_alloc_fast_reg_mr(fpo->fpo_hdev->ibh_pd,
1582 »·······»·······»·······»·······»·······»·······   LNET_MAX_IOV);
1583 #else
1584 »·······»·······/*
1585 »·······»······ * it is expected to get here if this is an MLX-5 card.
1586 »·······»······ * MLX-4 cards will always use FMR and MLX-5 cards will
1587 »·······»······ * always use fast_reg. It turns out that some MLX-5 cards
1588 »·······»······ * (possibly due to older FW versions) do not natively support
1589 »·······»······ * gaps. So we will need to track them here.
1590 »·······»····· */
1591 »·······»·······frd->frd_mr = ib_alloc_mr(fpo->fpo_hdev->ibh_pd,
1592 #ifdef IB_MR_TYPE_SG_GAPS
1593 »·······»·······»·······»·······»·····  ((*kiblnd_tunables.kib_use_fastreg_gaps == 1) &&
1594 »·······»·······»·······»·······»·····   (dev_caps & IBLND_DEV_CAPS_FASTREG_GAPS_SUPPORT)) ?
1595 »·······»·······»·······»·······»·····»·····IB_MR_TYPE_SG_GAPS :
1596 »·······»·······»·······»·······»·····»·····IB_MR_TYPE_MEM_REG,
1597 #else
1598 »·······»·······»·······»·······»·····»·····IB_MR_TYPE_MEM_REG,
1599 #endif
1600 »·······»·······»·······»·····  LNET_MAX_IOV);
1601 »·······»·····if ((*kiblnd_tunables.kib_use_fastreg_gaps == 1) &&
1602 »·······»·····    (dev_caps & IBLND_DEV_CAPS_FASTREG_GAPS_SUPPORT))
1603 »·······»·····»·····CWARN("using IB_MR_TYPE_SG_GAPS, expect a performance drop\n");
1604 #endif
1605 »·····»·····if (IS_ERR(frd->frd_mr)) {
1606 »·····»·····»·····rc = PTR_ERR(frd->frd_mr);
1607 »·····»·····»·····CERROR("Failed to allocate ib_fast_reg_mr: %d\n", rc);
1608 »·····»·····»·····frd->frd_mr = NULL;
1609 »·····»·····»·····goto out_middle;
1610 »·····»·····}
```

Active Connection Establishment

Once the ground work is laid down, then the LND waits for requests to do RDMA operations or for remote connections. The former is called Active Connection Establishment. This section will give an overview of how that works in the code. The latter is called Passive Connection Establishment and will be described in the following section.

When an RDMA operation is requested by higher up layers, an IOV is passed to the LND. The LND needs to map the memory to be RDMAed in preparation for posting. The maximum RDMA operation size the LND does is 1MB, broken into 256 4K (page size on x86-64 systems) work requests.

The code can be followed here:

```
kiblnd_setup_rd_iov()
or
kiblnd_setup_rd_kiov()
```

Once the memory to be RDMAed is mapped properly (mapping depends on whether we use FMR or FastReg), then a connection establishments process commences.

Step 1: resolve address:

```
rc = rdma_resolve_addr(cmid,
                                        (struct sockaddr *)&srcaddr,
                                        (struct sockaddr *)&dstaddr,
                                        lnet_get_lnd_timeout() * 1000);
```

Once we receive RDMA_CM_EVENT_ADDR_RESOLVED we proceed to step 2, resolve route:

```
rc = rdma_resolve_route(cmid, lnet_get_lnd_timeout() * 1000);
```

On RDMA_CM_EVENT_ROUTE_RESOLVED we move to step 3, create cq and the qp

```
 872 #ifdef HAVE_IB_CQ_INIT_ATTR
 873 »·······cq_attr.cqe = IBLND_CQ_ENTRIES(conn);
 874 »·······cq_attr.comp_vector = kiblnd_get_completion_vector(conn, cpt);
 875 »·······cq = ib_create_cq(cmid->device,
 876 »·······»·······»·······  kiblnd_cq_completion, kiblnd_cq_event, conn,
 877 »·······»·······»·······  &cq_attr);
 878 #else
 879 »·······cq = ib_create_cq(cmid->device,
 880 »·······»·······»·······  kiblnd_cq_completion, kiblnd_cq_event, conn,
 881 »·······»·······»·······  IBLND_CQ_ENTRIES(conn),
 882 »·······»·······»·······  kiblnd_get_completion_vector(conn, cpt));
 883 #endif

 898 »·······rc = ib_req_notify_cq(cq, IB_CQ_NEXT_COMP);

 904 »·······init_qp_attr->event_handler = kiblnd_qp_event;
 905 »·······init_qp_attr->qp_context = conn;
 906 »·······init_qp_attr->cap.max_send_sge = *kiblnd_tunables.kib_wrq_sge;
 907 »·······init_qp_attr->cap.max_recv_sge = 1;
 908 »·······init_qp_attr->sq_sig_type = IB_SIGNAL_REQ_WR;
 909 »·······init_qp_attr->qp_type = IB_QPT_RC;
 910 »·······init_qp_attr->send_cq = cq;
 911 »·······init_qp_attr->recv_cq = cq;
 912
 913 »·······conn->ibc_sched = sched;
 914
 915 »·······do {
 916 »·······»·······init_qp_attr->cap.max_send_wr = kiblnd_send_wrs(conn);
 917 »·······»·······init_qp_attr->cap.max_recv_wr = IBLND_RECV_WRS(conn);
 918
 919 »·······»·······rc = rdma_create_qp(cmid, conn->ibc_hdev->ibh_pd, init_qp_attr);
 920 »·······»·······if (!rc || conn->ibc_queue_depth < 2)
 921 »·······»·······»·······break;
 922
 923 »·······»·······conn->ibc_queue_depth--;
 924 »·······} while (rc);
```

The LND has its own protocol, where some messages are exchanged to determine the size of the RDMA that's about to happen. Once that's determined, then when initialize the RDMA operation.

```
wrq->wr.next»···= &(wrq + 1)-
>wr;
wrq->wr.wr_id»··= kiblnd_ptr2wreqid(tx,
IBLND_WID_RDMA);
wrq->wr.sg_list»=
sge;
wrq->wr.opcode»·=
IB_WR_RDMA_WRITE;
wrq->wr.send_flags = 0;

/* kiblnd_init_rdma() for more details */
```

We then post the work request on the qp

```
rc = ib_post_send(conn->ibc_cmid->qp, wr, &bad);
```

Note that the LND never does an RDMA read. It only does an RDMA write. This is for historical limitations, which might not be applicable with the latest technology.

Passive Connection Establishment

When the LND receives RDMA_CM_EVENT_CONNECT_REQUEST it proceeds to create the passive side of the connection. Basically it creates the CQ and QP as shown here.

Receiving RDMA

When a LND connection is created a number of buffers, each is 4K in size, are posted to the QP to receive incoming RDMAs. Receiving and sending messages in the LND is governed by a credit system to ensure the peers don't over flow the buffers on the QP.

Notes on RDMA and QP Timeouts