OPENFABRICS ALLIANCE

10TH ANNUAL INTERNATIONAL DEVELOPER WORKSHOP

OFVWG: RDMA CQ abstraction

Sagi Grimberg

CQ API

- Allocate a CQ

```
struct ib_cq *ib_create_cq(struct ib_device *device,
                           ib_comp_handler comp_handler,
                           void (*event_handler)(struct ib_event *, void *),
                           void *cq_context,
                           const struct ib_cq_init_attr *cq_attr)


static inline int ib_req_notify_cq(struct ib_cq *cq,
                                   enum ib_cq_notify_flags flags)
```

- Free a CQ

```
int ib_destroy_cq(struct ib_cq *cq)
```

CQ API

- Handling a CQ event

```
static void cq_event_handler(struct ib_cq *cq, void *cq_context)
{
     struct my_cq_context ctx = cq_context;

     tasklet_schedule(&ctx->taskslet);      /* Or */
     queue_work(&ctx->work);              /* Or */
     wake_up(ctx->thread)                 /* Or */
     /* poll from hard-IRQ context */
}
```

How do we handle the CQ?

- Which completion context is correct?
  - Kthread?
  - Workqueue?
  - Soft-IRQ?
  - Hard-IRQ? (please no!)
- How do I maintain fairness between CQs?
  - In cases where more than one CQ is assigned to the same MSIX
- How do I correctly re-arm my CQ?
  - Can I always re-arm my CQ?
  - What are MISSED_EVENTS?

How do we handle the CQ?

- How do I get my WR context back?
  - Is the wr_id always reliable?
  - Can I distinguish between two post_send completions (send/rdma/fastreg)?
- What is the best way to poll the CQ?
  - Batch polling?
  - 1-by-1?
- What are the affinity considerations?
  - Can I make sure of CPU/NUMA locality?

CQ polling API

- Polling CQ

```
static void my_cq_polling(struct ib_cq *cq, int budget)
{
        while ((n = ib_poll_cq(cq, 1, &wc)) > 0) {
        handle_wc(&wc);

        if (++completed >= budget)
                        break;
        }
    ib_req_notify_cq(cq, IB_CQ_NEXT_COMP);
}
```

- Usually handle_wc will look at the wc.status, wc.opcode and wc.wr_id to decide on what to do with the completion.

Handle WC

```
static void handle_cq(struct ib_wc *wc)
{
     struct my_ctx = wc->qp->qp_context;

     if (wc.status == IB_WC_SUCESS) {
          if (wc.opcode == IB_WR_SEND) {
               send_ctx = wc.wr_id;
               handle_send_comp(send_ctx);
          } else if (wc.opcode == IB_WR_RECV) {
               recv_ctx = wc.wr_id;
               handle_recv_comp(recv_ctx);
          } else if (wc.opcode == IB_WR_RDMA_WRITE) {
               rdma_w_ctx = wc.wr_id;
               handle_rdma_w_comp(rdma_w_ctx);
          }
          …
     } else {
          /* Who knows what to do… opcode field is not reliable */
          /* We do all sorts of hacks to understand the failure semantics */
     }
}
```

Proposal – CQ abstraction

- Get as much ULP code duplication into the core
- Handle correctly:
  - CQ re-arming
  - Polling contexts
  - Fairness and budgets
- Resolve the error completions un-reliability
- Optimize performance
  - Do the needed optimization magics
- Get it right once!

CQ abstraction API

- Allocate a CQ – select your polling context

```
enum ib_poll_context {
        IB_POLL_DIRECT                  = 1
        IB_POLL_SOFTIRQ                 = 2, /* poll from softirq context */
        IB_POLL_THREAD                  = 3, /* poll from thread context */
        IB_POLL_WORKQUEUE               = 4, /* poll from workqueue */
};

struct ib_cq *ib_alloc_cq(struct ib_device *dev, void *private,
                int nr_cqe, int comp_vector, enum ib_poll_context poll_ctx)
```

- Free a CQ

```
void ib_free_cq(struct ib_cq *cq)
```

- Don’t poll, don’t handle events, don’t maintain logic on what to do when…

WR and WC API

- Don’t poll, don’t handle events, don’t maintain logic on what to do when…
  - Just pass in a “done” routine handler

```
struct ib_cqe {
        void (*done)(struct ib_cq *cq,
                struct ib_wc *wc);
};
```

```
struct ib_send_wr {
        union {
                u64             wr_id;
                struct ib_cqe   *wr_cqe;
        };
    …
};
```

```
struct ib_recv_wr {
        union {
                u64             wr_id;
                struct ib_cqe   *wr_cqe;
        };
    …
};
```

```
struct ib_wc {
        union {
                u64             wr_id;
                struct ib_cqe   *wr_cqe;
        };
    …
};
```

Just fill in your “done”

- The core will do the actual polling and CQ handling, and will dispatch to the “done” handlers

```
static int ib_process_cq(struct ib_cq *cq, int budget)
{
        int i, n, completed = 0;

        while ((n = ib_poll_cq(cq, IB_POLL_BATCH, cq->wc)) > 0) {
                for (i = 0; i < n; i++) {
                        struct ib_wc *wc = &cq->wc[i];

                        if (wc->wr_cqe)
                                wc->wr_cqe->done(cq, wc);    /* This is the dispatch */
                        else
                                WARN_ON_ONCE(wc->status == IB_WC_SUCCESS);
                }

                completed += n;
                if (completed >= budget)
                        break;
        }

        return completed;
}
```

ULP will do…

- The ULP just make sure to have a per-WR type “done” handler

```
int send_message(struct send_ctx *ctx)
{
      struct ib_send_wr *wr;

      ctx->send_cqe.done = my_send_done;
      wr.wr_cqe = &ctx->send_cqe;
      return post_send(ctx->qp, wr);
}


void my_send_done(struct ib_cq *cq, struct ib_wc *wc)
{
      struct send_ctx *ctx = container_of(wc->wr_cqe);

      /* do … */
}
```

- Same for RECV, RDMA, MEM-REG …

Thank You

OPENFABRICS ALLIANCE

10TH ANNUAL INTERNATIONAL DEVELOPER WORKSHOP