Design Challenges of User-Level Protocols

By: Chethan K Rudramuni
Presentation Overview

- Why User-level Protocols?
- Discuss two systems
  - Myrinet System.
  - Gigabit Ethernet System.
- Design choices and challenges.
- Experiment results.
- Infiniband Verbs Implementation.
Motivation and Overview

- Latency = (Network Latency) + (Packet Processing Time)

- Modern Networks have reduced Network latency hence shifting focus on to time spent in packet processing.

- Traditional network protocols process messages in kernel which involve interrupts, multiple data copies etc.

- Modern NICs have programmable units that can be used to offload some part of processing from host, hence improving throughput.
Experimental Set-ups

Myrinet System
Gigabit Ethernet Experimental System

- Proposed Design showing offloading of processing from kernel to NIC.
Advantages of User-level Protocol

- OS bypass leads to lower latency and better throughput.
- Frees host CPU cycles for application.
- Makes use of NIC resources to handle processing.
Challenges in designing Userlevel protocols

- Data transfer mechanism.
- Virtual Memory Management.
- Framing and Reliability.
- Protection.
- Control Transfer.
- Recovering and Preventing Overflows.
- Multicasting.
Data Transfer

Host to NIC:
- We have choice between Programmed I/O and DMA.
  - Programmed I/O is generally slow and increases IO-channel traffic and uses Host CPU.
  - DMA is faster but has some start-up latency, requires address translation etc.

- Which is better?
  - Depends on entire system, but generally with CPU features like write-combining buffers, Programmed I/O can perform better than DMA for smaller messages.
Data Transfer

Interface to Host:
- DMA is generally better choice over PIO as IO-bus are very slow for read operations.
- Some implementations use PIO for smaller messages and DMA for larger ones.

To buffer or not?
- Buffering is limited by the amount of memory available in NIC.
- Buffering would result in multiple copies.
- EMP design takes a radical approach of dropping packets if there is no pre-posted receive.
Virtual Memory Management

● Using DMA has following problems.
  ○ DMA needs physical memory address, but application has virtual memory address, hence some address translation is required.
  ○ OS could potentially swap out memory page being used hence corrupting message, this should be prevented.

● Solutions:
  ○ Use Programmed IO.
  ○ Pin pages so they are not swapped out. (Pre-pinned or dynamically pinned)
  ○ Use kernel module for address translation.

● EMP solves this by mandating application to pass physical address and locking entire address space using mlockall.
Protocol Processing

Framing:
- EMP doesn't do any buffering to avoid overhead.
  - Send: NIC pulls data from host one frame at a time and sends it.
  - Receive: Accepts packets for pre-posted receives, and drops all other packets.

Reliability:
- EMP chooses to acknowledge collection of frames instead for each frame to avoid overhead.
Protection

- In the Myrinet example given, multi-users are not allowed as users could potentially corrupt each other's data. This is not desirable!

Solution:
- Allocate different parts of memory to different user hence avoiding conflict.
  - But this is limited by limited memory.
- Use paging concept, NIC can move inactive end-points to host memory and bring it back when required.
Control Transfer

- Interrupts are expensive hence use polling. NIC would set a flag and host keeps checking it.
  - Wastes host CPU cycles.
  - Would increase IO channel traffic.

- In multi-core systems, host could dedicate one of the core for checking flag and packet processing.
Recovering and Preventing Overflows

Myrinet:

- Some myrinet systems use ACKs and NACKs to signal status to the sender. But increases load.
- As myrinet is very reliable network, most of the packet loss are because of NIC dropping packets.
- Different systems use different flow control mechanisms.

EMP:

- It signals status for collection of packets instead of each packet.
- It drops packet if there is no pre-posted receive, avoiding buffering.
Multicast

- Naive approach of multicast with many point-to-point sends is highly inefficient.
- NIC could be programmed to carryout multicast at host and in the forwarding path is more efficient.
## Different Myrinet Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Data transfer (host-NI)</th>
<th>Translation</th>
<th>Protection</th>
<th>Control transfer</th>
<th>Reliability</th>
<th>Multicast support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM-II</td>
<td>PIO &amp; DMA</td>
<td>DMA areas</td>
<td>Yes</td>
<td>Polling + interrupts</td>
<td>Reliable, network interface: alternating bit, host: sliding window</td>
<td>No</td>
</tr>
<tr>
<td>FM</td>
<td>PIO</td>
<td>DMA area (recv)</td>
<td>No</td>
<td>Polling</td>
<td>Reliable, host-level credits</td>
<td>No</td>
</tr>
<tr>
<td>FM/MC</td>
<td>PIO</td>
<td>DMA area (recv)</td>
<td>No</td>
<td>Polling + interrupts</td>
<td>Reliable, unicast: host-level credits, multicast: network-interface-level credits</td>
<td>Yes</td>
</tr>
<tr>
<td>PM</td>
<td>DMA</td>
<td>Software TLB* on network interface</td>
<td>Yes</td>
<td>Polling</td>
<td>Reliable, ACK/NACK protocol on network interface</td>
<td>Yes (multiple sends)</td>
</tr>
<tr>
<td>VMMC</td>
<td>DMA</td>
<td>Software TLB on network interface</td>
<td>Yes</td>
<td>Polling + interrupts</td>
<td>Reliable, exploits hardware backpressure</td>
<td>No</td>
</tr>
<tr>
<td>VMMC-2</td>
<td>DMA</td>
<td>UTLB* in kernel, cached on network interface</td>
<td>Yes</td>
<td>Polling + interrupts</td>
<td>Reliable</td>
<td>No</td>
</tr>
<tr>
<td>LFC</td>
<td>PIO</td>
<td>User translates</td>
<td>No</td>
<td>Polling + interrupts + watchdog</td>
<td>Reliable, unicast: network-interface-level credits, multicast: network-interface-level credits</td>
<td>Yes (on network interface)</td>
</tr>
<tr>
<td>Hamlyn</td>
<td>PIO &amp; DMA</td>
<td>DMA areas</td>
<td>Yes</td>
<td>Polling + interrupts</td>
<td>Reliable, exploits hardware backpressure</td>
<td>No</td>
</tr>
<tr>
<td>Trapeze</td>
<td>DMA</td>
<td>DMA to page frames</td>
<td>No</td>
<td>Polling + interrupts</td>
<td>Unreliable</td>
<td>No</td>
</tr>
<tr>
<td>BIP</td>
<td>PIO &amp; DMA</td>
<td>User translates</td>
<td>No</td>
<td>Polling</td>
<td>Reliable, rendezvous and backpressure</td>
<td>No</td>
</tr>
<tr>
<td>U-Net</td>
<td>DMA</td>
<td>TLB on network interface (U-Net/MM)</td>
<td>Yes</td>
<td>Polling + interrupts</td>
<td>Unreliable</td>
<td>No</td>
</tr>
</tbody>
</table>
Myrinet Throughput

Throughput in host to interface transfer with different transfer mechanisms
Latency and Bandwidth Comparison as function of message size (in KB)
EMP Throughput Result

Throughput as function of CPU utilization (for 10KB messages)
Infiniband

● What is infiniband?
  ○ A comprehensive specification from physical to application layer, with high bandwidth and low latency as main focus.

● An application centric design with following features.
  ○ OS-bypass.
  ○ Hardware Based Transport Protocol.
  ○ RDMA-read and RDMA-write.
Verbs

- Verbs are interfaces to Channel Adapter.
- Not APIs, but interfaces that can be used to implement APIs.

Verb groups
- Transport Resource Management
  - HCA Access, Protection domain management, QP sanctions, memory management etc.
- Work Request Processing.
- Multicast Services for UD QPs.
- Event Notification and Handling.
Verb groups and Relationships
const size_t SIZE = 1024;

char *buffer = malloc(SIZE);
struct ibv_mr *mr;
uint32_t my_key;
uint64_t my_addr;

mr = ibv_reg_mr( pd, buffer, SIZE, IBV_ACCESS_REMOTE_WRITE);
my_key = mr->rkey;
my_addr = (uint64_t)mr->addr;

/* Send keys to Node-2 */

Node-2

char *buffer = malloc(SIZE);
struct ibv_mr *mr;
struct ibv_sge sge;
struct ibv_send_wr wr, *bad_wr;

mr = ibv_reg_mr( pd, buffer, SIZE, IBV_ACCESS_LOCAL_WRITE);

/*get peer_key and peer_addr from node-1 */
strcpy(buffer, "RDMA");

sge.addr = (uint64_t)buffer; sge.length = SIZE;
sge.lkey = mr->lkey;

wr.sg_list = &sge; wr.num_sge = 1;
wr.opcode = IBV_WR_RDMA_WRITE;
wr.wr.rdma.remote_addr = peer_addr;
wr.wr.rdma.rkey = peer_key;

ibv_post_send(qp, &wr, &bad_wr);
RDMA Example

● Node-1 registers its memory using memory management verb `ibv_reg_mr()` to get R_key and sends it to Node-2.

● Node-2 registers its buffer with HCA, writes some data.

● Node-2 gets R_key from Node-1 and posts work-request wr using send verb `ibv_post_send`.

● Prototypes of send and Registration verbs are given below.
  ○ `int ibv_post_send(struct ibv_qp *qp, struct ibv_send_wr *wr, struct ibv_send_wr **bad_wr);`
  ○ `struct ibv_mr *ibv_reg_mr(struct ibv_pd *pd, void *addr, size_t length, int access);`
  ■ Note that protection domain needs to be provided while registering memory.
Memory Management Verbs

● As in previous example, to do operations like RDMA without involving host CPU, we should be able to register memory with HCA and let it do DMA operation on this region.

● Accomplished by verbs given below:
  ○ struct ibv_mr *ibv_reg_mr(struct ibv_pd *pd, void *addr, size_t length, int access);
  ○ int ibv_dereg_mr(struct ibv_mr *mr);

● Required inputs:
  ○ Protection domain handle.
  ○ Address that needs to be registered.
  ○ Length of the memory region registered.
  ○ Access control (Local and remote access)

● Return value of type ibv_mr* from ibv_reg_mr will have
  ○ L_KEY
  ○ R_KEY (Optional)
Send/Receive Verbs

● Similar to RDMA, there is a 2-sided send/receive communication semantic.

● Send Receive verbs:
  ○ int `ibv_post_send` (struct `ibv_qp` *qp, struct `ibv_send_wr` *wr, struct `ibv_send_wr` **bad_wr);
  ○ int `ibv_post_recv` (struct `ibv_qp` *qp, struct `ibv_recv_wr` *wr, struct `ibv_recv_wr` **bad_wr);

  ■ `qp` - Queue pair handle.
  ■ `wr` - Null terminated list of Work Requests.
  ■ `bad_wr` - Output parameter that would point to the work request that failed.

● The HCA driver will convert `wr` into internal WQE format.
● Once posted, HCA is notified by writing into doorbell space.
QP verbs

● QPs is the way OS-bypass is done in infiniband.

● QP verbs:
  ○ struct ibv_qp* ibv_create_qp( struct ibv_pd *pd, struct ibv_qp_init_attr *qp_init_attr );
  ○ int ibv_modify_qp( struct ibv_qp *qp, struct ibv_qp_attr *attr, int attr_mask );
  ○ int ibv_destroy_qp(struct ibv_qp *qp);
  ■ pd - Protection domain.
  ■ qp_init_attr - Initial attributes like context, Queue type, WQE depth etc.
  ■ attr and attr_mask - Give required attribute values.

● Generally, attributes control the properties of QP like WQE depth, CQ, QP signalling type, Protection domain, type of QP etc.
Completion Queue Management Verbs

- Each send/receive queue pair will have an associated CQ
- CQs are the means for a verb consumer to obtain completion information.

CQ Management verbs:
- struct ibv_cq* ibv_create_cq(struct ibv_context *context, int cqe, void *cq_context, struct ibv_comp_channel *channel, int comp_vector);
  - context - Device context.
  - cqe - Minimum length of queue.
  - cq_context - Used to set user context field of CQ structure.
  - comp_vector - Signaling completion events.

- Resizing CQ:
  - Application might need to resize the CQ.
  - Should be done without any loss of completion records.
    - Create new CQ with new requested size.
    - VPD copies all the content into new CQ.
    - Destroy old CQ.
Completion Queue Management Verbs

- **CQ resize verb:**
  - int `ibv_resize_cq`(struct `ibv_cq` *cq, int cqe);
    - `cq` - CQ handle.
    - `cqe` - required size.
  - Previously given handle to CQ remains valid for the resized CQ.

- **CQ Destroy Verb:**
  - int `ibv_destroy_cq`(struct `ibv_cq` *cq);
    - `cq` - Handle to CQ.

- **Destroying CQ:**
  - There shouldn't be any queues associated with this CQ (ref count = 0).
  - When CQ is recycled and given to new process, HCA should make sure that stale events are not delivered to new owner. The implementation is vendor specific.
Thank you