CS647 Modern Storage Systems

- RDMA

Giorgos Saloustros, gesalous@ics.forth.gr

CS647 Modern Storage Systems, Fall 2022
(Modern Key-Value Store Design and Implementation)
Angelos Bilas, bilas@csd.uoc.gr
Computer Science Department, University of Crete

Remote Direct Memory Access

- Remote
  - Transfers between nodes in a network
- Direct
  - No kernel involvement in the data path
  - Reliable delivery implemented from network card/switch
  - Layer 2 protocols: Infiniband, RoCE
- Memory
  - Access through application virtual memory
  - No extra copy/buffering
- Access
  - send(), receive(), read(), write(), atomics
- Benefits: High throughput, low latency/CPU utilization, high messaging rate

RDMA Architecture

Verbs API

- InfiniBand specification written in terms of verbs
  - Semantic description of required behavior
  - No syntactic or operating system specific details
  - Implementations free to define their own API
- OpenFabrics Alliance (OFA) Verbs API
  - one possible syntactic definition of an API
  - in syntax, each “verb” becomes an equivalent “function”
  - done to prevent proliferation of incompatible definitions
  - was an OFA strategy to unify InfiniBand market

RDMA vs. TCP

- Both utilize the client-server model
- Both require a connection for reliable transport
- Both provide a reliable transport mode
  - TCP provides a reliable in-order sequence of **bytes**
  - RDMA provides a reliable in-order sequence of **messages**
- "zero copy" – data transferred directly from virtual memory on one node to virtual memory on another node
- "kernel bypass" – no operating system involvement during data transfers
- asynchronous operation – threads not blocked during I/O transfers

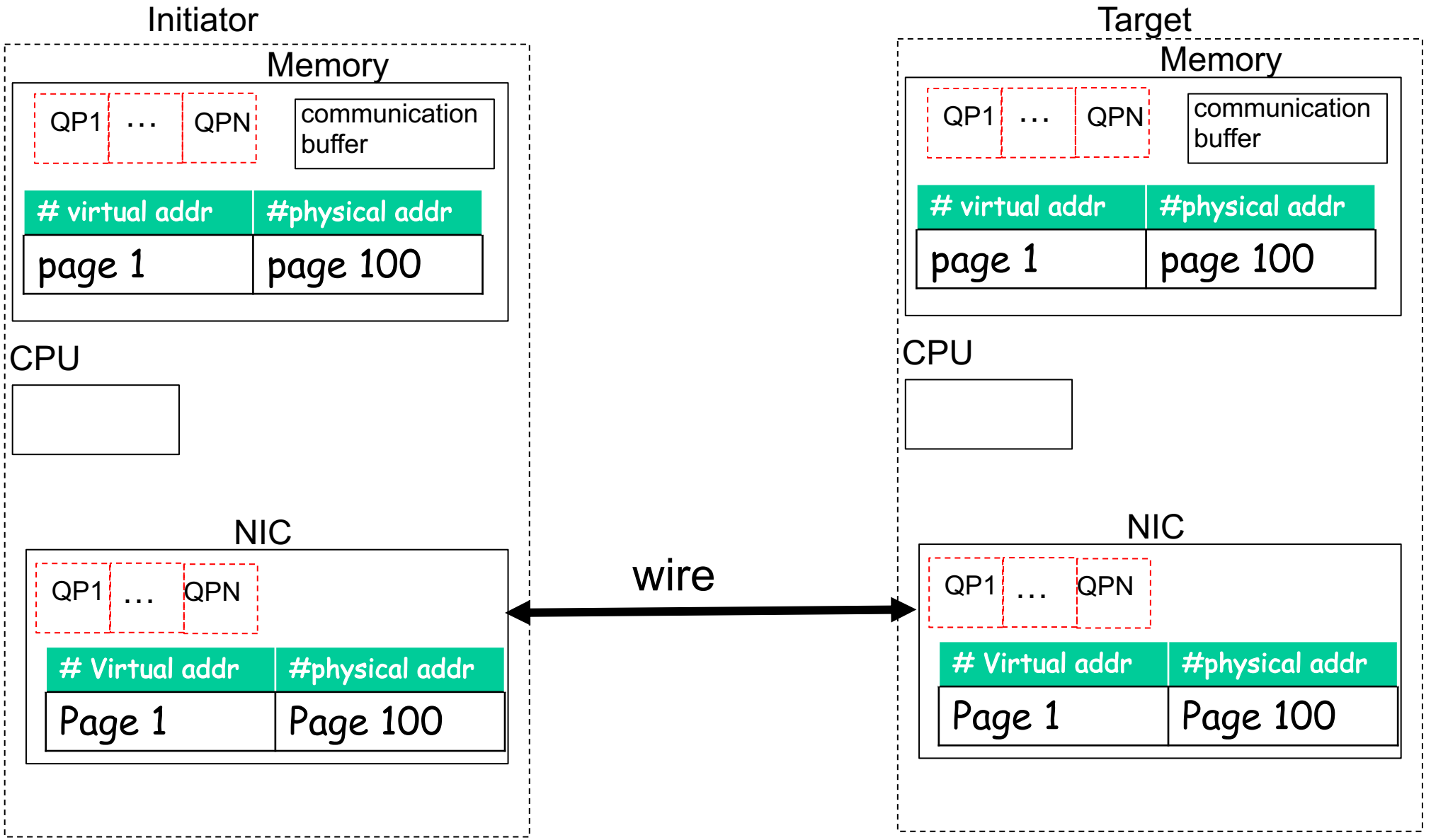
TCP/IP Access Model

- Byte streams – requires application to delimit / recover message boundaries
- Synchronous – blocks until data is sent/received
- send() and recv() are paired
  - Both CPUs involved in communication
- send() and recv() are paired
- Requires data copy into system buffers

RDMA Access Model and Operations

- Messages – preserves user's message boundaries
- Asynchronous – no blocking during a transfer, which
  - Start on posting a Work Request
  - Finish on receiving a completion
- 2-sided
  - send(), recv() both CPUs involved in communication – no system copies
- 1-sided
  - read(), write(): Target's CPU not involved in communication
- Atomics: (Infiniband, RoCE)

RDMA Architecture



Receive Path: RDMA vs TCP

Posting-Completion

Asynchronous Data Transfer

- Posting
  - Adds a work request (WR) to the work queue of CA
  - WR describes the desired operation
- Completion
  - Marks the end of a data transfer
  - Application removes a work completion from completion queue
- Take place in user space, no Kernel involvement
- During transfer application *must not* touch the virtual memory area involved
  - Otherwise, undefined behaviour

RDMA recv and Completion Queues

RDMA send() / recv()

- Connection establishment similar to TCP/IP
  - listen(), accept()
- Sender and Receiver CPUs are involved
  - Sender issues send() operations
  - Receiver issues recv() operations
- Both parties do not know a priori the receiver's/sender's memory location

Basic Structures

- Queue Pair (QP): communication endpoint
  - Consists of two queues: Send and Receive
- Communication buffers consist of virtual addresses
- QP and virtual address mappings must be in the NIC

Where to Write and How to Receive?

- Where in the target's memory should initiator write to?
  - Destination agnostic (two-sided operation)
  - Destination known (one-sided operation)
- Two sided operations: Target registers a buffer prior to a send()
  - No dedicated communication buffer / multiplex buffers between clients
  - System call per buffer (or in batches)
  - Message max size must be known a priori up to a MAX_SIZE
- One sided operations: Direct read/write from remote memory
  - No CPU involvement of target
  - Dedicated buffer per client
- Detect messages via polling (in user space)
  - Optimal latency (order of us)
  - Consumes constantly CPU cores even at low utilization
- Detect messages via interrupts from the completion queue
  - No cores spinning
  - Latency and CPU overhead from interrupts

RDMA Scaling with #Clients

- Need to scale up to million clients
  - Data grow exponentially
  - HPC computations need tenths of thousands of nodes / Cloud
- Two-sided operations
  - 100 Gbps ~= 12.8 GB/s, message size ~= 256 B, 53 M messages/s
  - Modern NICs are capable of 100 M packets/s
  - Can we saturate link throughput ?
- One-sided operations
  - Communication buffer 256 KB / client ~= 256 GB
  - Mappings (assuming 16 B / entry) ~= 1 GB

Must be in NIC's memory

Scale Queue Pairs (QP)

- Assume QP are of size of ~= 128 B
- For 1 M QP we need 128 MB state in the NIC
  - For 10 M → 1.2 GB (probably OK with modern NICs)
- NIC evict QPs in DRAM
  - Introduce page faults in the NIC
- Use unreliable queue pairs
  - No FIFO order, messages can be lost
  - Application must implement this functionality
- Use a set of TCP/IP proxies
  - Clients interact with TCP/IP proxies
  - Proxies interact via RDMA with servers
- Mutliplex QPs inside the kernel
  - Introduction of kernel in the data path

Scaling of Polling

- How to check 256 GB of communication buffers?
- Vast address space / no cache locality
- Split communication buffers into regions
  - Poll via separate threads
  - Consume cycles
- How many threads?
  - Use N threads → Each thread uses internally an FSM multiplex requests