iWarp-Protocol Kernel-Space Software Implementation

Dennis Dalessandro, Ananth Devulapalli, Pete Wyckoff

{dennis, ananth, pw}@osc.edu

Ohio Supercomputer Center
Overview

- Introduction
- Motivation: Why Software iWarp?
- iWarp: Details
- Implementation details
- Experiments & Results
- Our future goals
Introduction

- High Performance Interconnects
  - Zero-copy
  - RDMA
  - Specialty protocol
  - LAN-wide

- RDMA over Ethernet → *iWarp*
  - De-congest data-path at the end-points
  - 10 GBps at 3-4 GHz
Motivation

- Single-sided Acceleration
- Flexible Research Platform
- Advantages of iWarp in kernel
Motivation

- Single-sided Acceleration
  - Hardware-enabled Server, Software-enabled Clients
  - Performance penalty at software end 😞
  - Hardware Accelerated server 😊
  - Cost-effective intermediate step

- Flexible Research Platform

- Advantages of iWarp in kernel
Motivation

- Single-sided Acceleration
  - Hardware-enabled Server, Software-enabled Clients
  - Performance penalty at software end 😞
  - Hardware Accelerated server 😊
  - Cost-effective intermediate step

- Flexible Research Platform
  - Protocol Experimentation
  - Protocol Compliance
  - Extensible to other protocols: *iSER, SRP*

- Advantages of iWarp in kernel
Motivation

- Single-sided Acceleration
  - Hardware-enabled Server, Software-enabled Clients
  - Performance penalty at software end 😞
  - Hardware Accelerated server 😊
  - Cost-effective intermediate step

- Flexible Research Platform
  - Protocol Experimentation
  - Protocol Compliance
  - Extensible to other protocols: iSER, SRP

- Advantages of iWarp in kernel
  - Unlock iWarp for kernel-resident clients: NFS
  - Coupling with TCP
  - Reduction in overhead
iWarp Details

**iWarp Stack**

- APPLICATION
  - VERBS
  - RDMAP
  - DDP
  - MPA
    - SCTP
  - TCP
  - IP
  - DATA LINK
  - PHYSICAL

April 20, 2006
Implementation Issues

- Verbs
- TCP Interface
- Threading Model
- Memory Registration Issues
Impl. Issues: Verbs

- Verbs or API like DAPL?
- User-space resident
- Character device interface with kernel module
- Modularized implementation
  - Single code-base for both user and kernel based implementations
- Minimize scope without sacrificing functionality
Impl. Issues: *TCP Interface*

- `kernel_sendmsg`, `kernel_recvmsg`
  - Blocked sends
  - Polling recvs
- MPA loosely coupled with TCP
  - *Flexibility versus Functionality*
Impl. Issues: Threading Model

- Multi-threading: non-blocking, asynchronous
- Single Threaded model
  - Simplicity versus Performance
Impl. Issues: Memory Registration

- Pre-registration of application buffers
- `kmap` and `kunmap`
- Book-keeping using reference counting
- 64-bit machines
- Overhead
About the code base

- iWarp software stack works against Ammasso 1100 RNIC
- CRC and Markers: switched on and off
- 20,000 lines of ANSI C code (user and kernel)
- Linux 2.6 kernel
- 32-bit and 64-bit support
Experimental Setup

- 71 node cluster with 41 Ammasso 1100 RNIC cards
  - Beta cards with FPGA-based IP
  - RDMA data-path and TCP data-path
- Dual Opteron 250 processors
  - One processor disabled for utilization tests
- 2GB RAM, 80GB SATA drives
- 2 Tigon Gigabit Ethernet NICs
- Tyan S2891 Motherboard
- 2 SMC switches
  - Switches introduce 2.8 µs latency
# Latency

<table>
<thead>
<tr>
<th></th>
<th>4 byte messages</th>
<th>64 kB messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>hw-hw</td>
<td>16.1 ± 0.3</td>
<td>614.2 ± 3.3</td>
</tr>
<tr>
<td>ksw-hw</td>
<td>18.7 ± 0.2</td>
<td>619.7 ± 1.2</td>
</tr>
<tr>
<td>tcp-tcp</td>
<td>16.9 ± 0.2</td>
<td>594.8 ± 18.9</td>
</tr>
</tbody>
</table>

Table 1: Latency overview (µs).

- Latency: 1/2-way ping-pong delay
- Back-to-back: bypass switch
- Small overhead
Throughput

- Sender and Receiver
- TCP > 10 MBps
Latency: *Kernel v/s User*

<table>
<thead>
<tr>
<th></th>
<th>4 B messages</th>
<th>64 kB messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel with CRC</td>
<td>20.3 ± 0.2</td>
<td>615.5 ± 1.2</td>
</tr>
<tr>
<td>user with CRC</td>
<td>19.6 ± 0.2</td>
<td>612.3 ± 1.9</td>
</tr>
<tr>
<td>kernel without CRC</td>
<td>20.1 ± 0.2</td>
<td>604.5 ± 0.8</td>
</tr>
<tr>
<td>user without CRC</td>
<td>19.5 ± 0.2</td>
<td>602.7 ± 0.8</td>
</tr>
</tbody>
</table>

Table 2: User vs kernel space latency (μs).

- CQ in kernel
- *kmap/kunmap* overhead
**Throughput: Kernel v/s User**

- CRC takes away 8 MBps
- Kernel and User space similar
CPU utilization Hardware

Figure 1: Hardware ↔ Hardware
- Subtractive method
- Hardware and Software Identical

Figure 2: Hardware ↔ Software
CPU utilization TCP and Software

Figure 3: TCP ↔ TCP
- TCP costly than Hardware iWarp
- Software is CPU intensive: CRC

Figure 4: Software ↔ Software
CPU Utilization without CRC

- 20% for Receiver and 40% Sender load due to CRC
- The loads in range of TCP
Related Work

- User-space software iWarp
- Sockets-based iWarp
- Other verbs: DAT Collaborative, OpenFabrics
Future Work

- Porting kernel space clients
- Integrating MPA with TCP
- WAN deployment
- iSER/SRP extensions
- Multithreaded stack
Conclusions

- Demonstrated interoperability with Hardware iWarp
- Demonstrated single-sided acceleration capability
- Software iWarp for kernel-resident clients
- Software iWarp is logical step before full deployment
Software Availability