Performance Isolation
Anomalies in RDMA

Yiwen Zhang
with Juncheng Gu, Youngmoon Lee, Mosharaf Chowdhury, and Kang G. Shin
Cloud operators are aggressively deploying RDMA in datacenters\cite{1}\cite{2}\cite{3}

\cite{1} Guo, Chuanxiong, et al. “RDMA over Commodity Ethernet at Scale.” SIGCOMM’16
\cite{2} Mittal, Radhika, et al. “TIMELY: RTT-based Congestion Control for the Datacenter.” SIGCOMM’15
\cite{3} Zhu, Yibo, et al. “Congestion control for large-scale RDMA deployments.” SIGCOMM’15
Cloud operators are aggressively deploying RDMA in datacenters\[1\][2][3]

Growing demands in ultra-low latency applications
- Key-value store & remote paging

High bandwidth applications
- Cloud storage & memory-intensive workloads

RDMA Is Being Deployed in Datacenters

Cloud operators are aggressively deploying RDMA in datacenters

RDMA provides both low latency and high bandwidth
  • Order-of-magnitude improvements in latency and throughput
  • With minimal CPU overhead!
At large-scale deployments, RDMA-enabled applications are unlikely to run in vacuum – the network must be shared.
Great! But There Are Limits . . .

At large-scale deployments, RDMA-enabled applications are unlikely to run in vacuum – the network must be shared.

HPC community uses static partitioning to minimize sharing.[1]

Researches in RDMA over Ethernet-based datacenters focus on the vagaries of Priority-based Flow Control (PFC).[2][3]

What Happens When Multiple RDMA-Enabled Applications Share The Network?
At A First Glance...

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td></td>
</tr>
</tbody>
</table>
At A First Glance...

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td></td>
</tr>
<tr>
<td>10B vs. 1MB</td>
<td></td>
</tr>
</tbody>
</table>
At A First Glance...

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td></td>
</tr>
<tr>
<td>10B vs. 1MB</td>
<td></td>
</tr>
<tr>
<td>1MB vs. 1MB</td>
<td></td>
</tr>
</tbody>
</table>
At A First Glance...

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td></td>
</tr>
<tr>
<td>10B vs. 1MB</td>
<td></td>
</tr>
<tr>
<td>1MB vs. 1MB</td>
<td></td>
</tr>
<tr>
<td>1MB vs. 1GB</td>
<td></td>
</tr>
</tbody>
</table>
Benchmarking Tool \[1\]

Modified based on *Mellanox Perftest* tool
- Creates 2 flows to simultaneously transfer a stream of messages
- Single queue pair for each flow
- Measures bandwidth and latency characteristics only when both flows are active

Benchmarking Tool

Modified based on *Mellanox Perftest* tool

- Creates 2 flows to simultaneously transfer a stream of messages
- Single queue pair for each flow
- Measures bandwidth and latency characteristics only when both flows are active
- Both flows share the same link

[1] https://github.com/In niswap/frdma_benchmark
RDMA Design Parameters

RDMA Verbs
- WRITE, READ, WRITE WITH IMM (WIMM), and SEND/RECEIVE

Transport Type
- All experiments using Reliable-Connected (RC) Queue Pairs

INLINE Message
- Enabled INLINE message for 10 Byte and 100 Byte messages in the experiment
Application-Level Parameters

Request Pipelining
• Provide better performance, but hard to configure for fair comparison
• Disabled by default

Polling mechanism
• Busy vs Event-triggered polling
Message Acknowledgement

- Next work request is posted until the WC of the previous one is polled from CQ
- No other flow control acknowledgment is used
Define an Elephant and a Mouse
Elephant vs. Elephant

Compare two throughput-sensitive flows by varying verb types, message sizes, and polling mechanism.

- WRITE, READ, WIMM, & SEND verbs transferring 1 MB & 1 GB messages
- Total amount of data transferred fixed at 1 TB
- Both flows using event-triggered polling
- Generated bandwidth ratio matrix
Elephant vs. Elephant: Larger Flows Win
Getting Better with Larger Base Flows

![Graph showing throughput ratio vs message size ratio]

- Throughput Ratio
- Message Size Ratio

- Throughput Ratio values: 0.750, 1.000, 1.250, 1.500
- Message Size Ratio values: 1, 2, 5, 10, 100, 1000
Getting Better with **Larger Base Flows**
Both flows use busy-polling.
But There Is a Tradeoff in CPU Usage

![Graph showing CPU Usage vs Message Size]

- Event-triggered
- Busying-polling

CPU Usage (%)

Message Size (Byte)
At A First Glance...

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td></td>
</tr>
<tr>
<td>10B vs. 1MB</td>
<td></td>
</tr>
<tr>
<td>1MB vs. 1MB</td>
<td></td>
</tr>
<tr>
<td>1MB vs. 1GB</td>
<td></td>
</tr>
</tbody>
</table>
At A First Glance...

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td></td>
</tr>
<tr>
<td>10B vs. 1MB</td>
<td></td>
</tr>
<tr>
<td>1MB vs. 1MB</td>
<td>Fair</td>
</tr>
<tr>
<td>1MB vs. 1GB</td>
<td></td>
</tr>
</tbody>
</table>
At A First Glance...

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td></td>
</tr>
<tr>
<td>10B vs. 1MB</td>
<td></td>
</tr>
<tr>
<td>1MB vs. 1MB</td>
<td>Fair</td>
</tr>
<tr>
<td>1MB vs. 1GB</td>
<td>Unfair</td>
</tr>
</tbody>
</table>
## At A First Glance...

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td></td>
</tr>
<tr>
<td>10B vs. 1MB</td>
<td></td>
</tr>
<tr>
<td>1MB vs. 1MB</td>
<td>Depends on CPU</td>
</tr>
<tr>
<td>1MB vs. 1GB</td>
<td>Unfair</td>
</tr>
</tbody>
</table>
At A First Glance...

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td></td>
</tr>
<tr>
<td>10B vs. 1MB</td>
<td></td>
</tr>
<tr>
<td>1MB vs. 1MB</td>
<td>Depends on CPU</td>
</tr>
<tr>
<td>1MB vs. 1GB</td>
<td>Depends on CPU</td>
</tr>
</tbody>
</table>
Mouse vs. Mouse: Pick a Base Flow

Compare two latency-sensitive flows with varying message sizes.

- All flows using WRITE operation with busy polling
- 10B, 100B and 1KB messages
- Pick 10B as base flow
- Measured latency and MPS of the base flow transferring 10 million messages at the presence of a competing flow
Mouse vs. Mouse: Worst Tails

- Latency (us)
  - Median
  - 99.99th
  - Million Messages/sec

<table>
<thead>
<tr>
<th></th>
<th>10B Alone</th>
<th>10B vs. (10B)</th>
<th>10B vs. (100B)</th>
<th>10B vs. (1KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (us)</td>
<td>1.3</td>
<td>1.3</td>
<td>1.4</td>
<td>1.3</td>
</tr>
<tr>
<td>Median</td>
<td>5.4</td>
<td>5.8</td>
<td>7.0</td>
<td>7.8</td>
</tr>
<tr>
<td>99.99th</td>
<td>0.78</td>
<td>0.76</td>
<td>0.72</td>
<td>0.75</td>
</tr>
</tbody>
</table>
### At A First Glance…

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td></td>
</tr>
<tr>
<td>10B vs. 1MB</td>
<td></td>
</tr>
<tr>
<td>1MB vs. 1MB</td>
<td>Depends on CPU</td>
</tr>
<tr>
<td>1MB vs. 1GB</td>
<td>Depends on CPU</td>
</tr>
</tbody>
</table>
At A First Glance…

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td>Good enough</td>
</tr>
<tr>
<td>10B vs. 1MB</td>
<td></td>
</tr>
<tr>
<td>1MB vs. 1MB</td>
<td>Depends on CPU</td>
</tr>
<tr>
<td>1MB vs. 1GB</td>
<td>Depends on CPU</td>
</tr>
</tbody>
</table>
Mouse vs. Elephant

Study performance isolation of a mouse flow running under a background elephant flow.

- All flows using WRITE operation
- All mouse flows sending 10 millions messages
- Mouse flows using busy polling while background elephant flows using event-triggered polling
- Measured latency and MPS of mouse flows
Mouse vs. Elephant: Mouse Flows Suffer

Latency (us)

- Median
- 99.99th

<table>
<thead>
<tr>
<th></th>
<th>10B Alone</th>
<th>10B vs. (1MB)</th>
<th>10B vs. (1GB)</th>
<th>100B Alone</th>
<th>100B vs. (1MB)</th>
<th>100B vs. (1GB)</th>
<th>1KB Alone</th>
<th>1KB vs. (1MB)</th>
<th>1KB vs. (1GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.3</td>
<td>5.4</td>
<td>8.1</td>
<td>9.2</td>
<td>5.5</td>
<td>8.5</td>
<td>9.5</td>
<td>6.0</td>
<td>6.3</td>
</tr>
<tr>
<td>Latency</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>12</td>
<td>16</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2.8</td>
<td>2.9</td>
<td>2.9</td>
<td>2.6</td>
<td>2.9</td>
<td>2.4</td>
<td>6.0</td>
<td>6.3</td>
<td>6.0</td>
</tr>
<tr>
<td></td>
<td>10B vs. (1MB)</td>
<td>10B vs. (1GB)</td>
<td>100B Alone</td>
<td>100B vs. (1MB)</td>
<td>100B vs. (1GB)</td>
<td>1KB Alone</td>
<td>1KB vs. (1MB)</td>
<td>1KB vs. (1GB)</td>
<td></td>
</tr>
</tbody>
</table>
Mouse vs. Elephant: Mouse Flows Suffer

- 0.79 Million Messages/sec for 10B Alone
- 0.36 Million Messages/sec for 10B vs. (1MB)
- 0.34 Million Messages/sec for 10B vs. (1GB)
- 0.71 Million Messages/sec for 100B Alone
- 0.39 Million Messages/sec for 100B vs. (1MB)
- 0.35 Million Messages/sec for 100B vs. (1GB)
- 0.42 Million Messages/sec for 1KB Alone
- 0.16 Million Messages/sec for 1KB vs. (1MB)
- 0.17 Million Messages/sec for 1KB vs. (1GB)
At A First Glance...

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td>Good enough</td>
</tr>
<tr>
<td>10B vs. 1MB</td>
<td></td>
</tr>
<tr>
<td>1MB vs. 1MB</td>
<td>Depends on CPU</td>
</tr>
<tr>
<td>1MB vs. 1GB</td>
<td>Depends on CPU</td>
</tr>
</tbody>
</table>
**At A First Glance...**

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Fair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10B vs. 10B</td>
<td>Good enough</td>
</tr>
<tr>
<td>10B vs. 1MB</td>
<td>Unfair</td>
</tr>
<tr>
<td>1MB vs. 1MB</td>
<td>Depends on CPU</td>
</tr>
<tr>
<td>1MB vs. 1GB</td>
<td>Depends on CPU</td>
</tr>
</tbody>
</table>
So far we ran all experiments using Mellanox FDR ConnectX-3 (56 Gbps) NIC on CloudLab.

Switch to Mellanox EDR ConnectX-4 (100 Gbps) NIC on the Umich Conflux cluster.

• The isolation problem in the elephant vs. elephant case still exists with a throughput ratio of 1.32.
• In the mouse vs. mouse case the problem appears to be mitigated; we did not observe large tail-latency variations when two mouse flows compete.
• In the mouse vs. elephant scenario, mouse flows are still affected by large background flows, where the median latency increases by up to 5×.
What Happens to Isolation in More Sophisticated and Optimized Applications?
Interested to know how isolation is maintained in HERD at a presence of a background elephant flow.

Running HERD on the Umich Conflux cluster.
- 5 million PUT/GET requests.
- Background flows using 1MB or 1GB messages with event-triggered polling
- Measured median and tail latency of HERD requests with and without a background flow

HERD vs. Elephant: HERD Also Suffers

Latency (us)

- GET Alone: 3.4, 9.5
- GET vs. (1MB): 15.9
- GET vs. (1GB): 26.9
- PUT Alone: 2.9
- PUT vs. (1MB): 14.5
- PUT vs. (1GB): 27.1

Median and 99.99th percentiles are indicated.
HERD vs. Elephant: Summary

HERD also has isolation issues when running with big background flows.

Currently, we are working on a solution to provide isolation in RDMA.

Special thanks to Yue Tan’s great help in generating isolation data on HERD.
Summary

- When the size difference of two flows are small, no matter they are small flows or very big flows, the isolation appears to be good.
- How fast an application can post RDMA requests onto the RNIC is the only thing that matters in a throughput-sensitive environment.
- When the size difference of two flows are big, there is a performance degradation of the smaller flow.
- Current hardware might not help to entirely resolve the issue.
Mouse Flow Latency

![Graph showing latency vs. message size for WRITE, READ, and SEND operations.](image)
Elephant vs. Elephant: Matrix

<table>
<thead>
<tr>
<th></th>
<th>SEND</th>
<th>WIMM</th>
<th>READ</th>
<th>WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMB</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>1GB</td>
<td>1.00</td>
<td>0.70</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>IMB</td>
<td>1.41</td>
<td>1.44</td>
<td>1.39</td>
<td>1.40</td>
</tr>
<tr>
<td>1GB</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>IMB</td>
<td>1.02</td>
<td>0.71</td>
<td>0.99</td>
<td>1.00</td>
</tr>
<tr>
<td>1GB</td>
<td>1.00</td>
<td>0.70</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>IMB</td>
<td>1.40</td>
<td>1.43</td>
<td>1.37</td>
<td>1.00</td>
</tr>
<tr>
<td>1GB</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMB</td>
<td>1.40</td>
<td>1.44</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>1GB</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mouse vs. Mouse: Unpredicted Behavior

Latency Ratio

MPS Ratio
HERD vs. Elephant: HERD Also Suffers

GET Alone

GET vs. (1GB Req)

GET vs. (1GB Resp)

GET vs. (1MB Req)

GET vs. (1MB Resp)

Latency (us)
HERD vs. Elephant: HERD Also Suffers

<table>
<thead>
<tr>
<th>Operation</th>
<th>1GB Req</th>
<th>1GB Resp</th>
<th>1MB Req</th>
<th>1MB Resp</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT Alone</td>
<td>2.9</td>
<td>7.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUT vs. (1GB Req)</td>
<td>12.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUT vs. (1GB Resp)</td>
<td>12.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUT vs. (1MB Req)</td>
<td>8.8</td>
<td>14.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUT vs. (1MB Resp)</td>
<td>7.6</td>
<td>18.8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Latency (us)
Summary

Elephant vs. Elephant:
- Polling mechanism dictates bandwidth allocation
- How fast an application can post RDMA requests onto the RNIC is the only thing that matters in a throughput-sensitive environment
- Tradeoff between CPU and Bandwidth

Mouse vs. Mouse:
- Little predictability between flows using equal-sized messages
- Increase in tail latency and decrease in MPS
- Isolation issue mitigated when switching to better hardware
Summary

Mouse vs. Elephant:
• In the presence of both types of flows, latency-sensitive flows suffer
• The requests posted by the mouse flows may queue up in RNIC’s queue buffer while the RNIC is doing continuous DMA reads from the main memory due to the background flow

HERD vs. Elephant:
• Isolation issues remain when running with background elephant flows
  Up to 4x increase in the median latency