RFP: When RPC is Faster than Server-Bypass with RDMA

Maomeng Su⁠¹, Mingxing Zhang⁠¹, Kang Chen⁠¹, Zhenyu Guo², Yongwei Wu⁠¹

¹Tsinghua University
²Microsoft Research
RDMA is a novel networking technology that offers low-latency, high-bandwidth, and server-bypassing features.

InfiniBand is one of the most popular hardware devices that supports RDMA.

- **Low-latency**: 1-3μs
- **High-bandwidth**: Up to 100Gb/s
- **Server-bypassing**: Server CPU and OS aware nothing about data transfer even the data is already in the server’s memory.
RDMA-based related work

- **Server-Reply**
  - RDMA-Memcached

- **Features**
  - * Using RDMA to replace TCP/IP
  - * Client send requests using RDMA write to Server
  - * Server return results using RDMA write to Client

- **Server-Bypass**
  - Pilaf [ATC 2013]
  - FaRM-KV [NSDI 2014]
  - C-Hint [SOCC 2014]

- **Features**
  - Clients access remote data structures by RDMA_Read/Write
  - Totally bypass Server CPU
  - Need great efforts
In-bound RDMA, Out-bound RDMA

Our Scenario: One server serves many clients using RDMA.
In-bound vs. Out-bound Asymmetry

The peak IOPS of in-bound (11.26MOPS) is about 5x higher than that of out-bound (2.11MOPS)

The chart illustrates the comparison between in-bound and out-bound RDMAs. The peak IOPS for in-bound is about 5 times higher than that of out-bound. The graph shows the number of server threads on the x-axis and IOPS (MOPS) on the y-axis.

In-bound Testing: RDMA Read

Out-bound Testing: RDMA Write
Observation 1

• In-bound vs. Out-bound Asymmetry

  * Overhead: Issuing RDMA > Serving RDMA

  → Performance of In-bound RDMA is better than Out-bound RDMA

  Limitation of server-reply mode
The Cost of CPU Bypass

* Server knows nothing and does nothing
* Clients need coordination
→ More RDMA operations

E.g., Pilaf uses 3.2 RDMA for 95% GET for read-intensive workloads
Worse for write-intensive workloads
Observation 2

- **Bypass Access Amplification**
  - No CPU processing on server
  - Clients need coordination
  - → Lead to more RDMA operation rounds
  - → Two roundtrips are not enough

Limitation of server-bypass mode
int GET(int server_id, void *key, int key_size, void *data_buf) {
    while(true){
        md=probe_metadata(server_id);
        while(true){
            data=get_data(s_id, md, data_buf);
            if checksum of data_buf is ok:
                break;
        }
        get key_size’ and value_size;
        if equal(key, key_size, data_buf, key_size’)
            break;
    }
    return value_size;
}

int GET(int server_id, void *key, int key_size, void *value_buf) {
    r_buf=prepare_request(key, key_size, GET_MODE);
    client_send(s_id, r_buf, sizeof(r_buf));
    size=client_recv(s_id, value_buf);
    return size;
}

Familiar with server-reply mode.

Special detection
Special data structures
## Design Choices for RPC system

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Programmability</th>
<th>Performance Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-Reply</td>
<td>Good</td>
<td>Limited by out-bound performance</td>
</tr>
<tr>
<td>Server-Bypass</td>
<td>Need great effort, Special data structures</td>
<td>Limited by number of retries</td>
</tr>
</tbody>
</table>

### RPC Phases

<table>
<thead>
<tr>
<th>RPC Phases</th>
<th>Request Send</th>
<th>Request Process</th>
<th>Result Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-reply</td>
<td>In-bound RDMA</td>
<td>Server involved</td>
<td>Out-bound RDMA</td>
</tr>
<tr>
<td>Server-bypass</td>
<td>In-bound RDMA</td>
<td>Server bypassed</td>
<td>In-bound RDMA</td>
</tr>
<tr>
<td><strong>RFP</strong></td>
<td>In-bound RDMA</td>
<td>Server involved</td>
<td>In-bound RDMA</td>
</tr>
<tr>
<td>Meaningless</td>
<td>In-bound RDMA</td>
<td>Server bypassed</td>
<td>Out-bound RDMA</td>
</tr>
</tbody>
</table>
RFP (Remote Fetching Paradigm) Overview

* Always use in-bound operations * Use server CPU to support RPC
* No client coordination * programmability is good
RFP (Remote Fetching Paradigm) Challenges

✓ **When** clients should fetch the results from server?

✓ **What size** for clients to fetch the results?
When Clients Fetch the Results in RFP

Continuously issuing RDMA_READ?
* Waste CPU cycles of clients
* Waste In-bound RDMA resources of server

RFP uses hybrid mechanism with a threshold $R$
* Continuously fetch $R$ times
* Switch to server-reply mode afterwards

$R$ is application and system specific
What size client should fetch?

F: the size for fetching results
Too large?
* Waste of network resources.
Too small?
* Need two fetches, first fetch contains the size.

RFP tries to avoid 2 fetches as much as possible.
* Application and system specific.
How much does RFP outperform server-reply and server-bypass?
How does RFP perform under different workloads and datasets?
How to choose R and F?
Evaluation

Set up

- A cluster of eight machines
- Dual eight-core CPUs (2.0 GHz), 96 GB RAM, Mellanox ConnectX-3 NIC (40 Gbps)
- Ubuntu 14.04
- Mellanox InfiniScale-IV switch

Datasets and Workloads

- Key-Value store
- Datasets: Uniform vs. Skew (Zipf distribution with parameter .99), generated by YCSB. 128 million keys (key size 8-byte).
- Workloads: Different GET percentiles (95%, 50%, 5%)
Datasets: Uniform datasets, different value sizes.
Workloads: 50% GET
Throughput: $RFP = 4x$ Pilaf’s.
Latency: RFP: 2 roundtrips, Pilaf: 3.2 roundtrips
Cannot support RPC
Using server CPU can support RPC

Use Two systems for comparison:

**ServerReply**: A simple implementation of key-value store (separate data structure)

**RDMA-Memcached**: Using RDMA to replace the communication (shared data structure)
Datasets: Uniform dataset, same value size (32B)
Workloads: 50% GET
Throughput: RFP = 2.6 x ServerReply
= 4.1 x RDMA-Memcached
Datasets: Uniform dataset, same value size (32B)
Workloads: 50% GET
Latency: RFP, 5.78 us
ServerReply: 12.06us,
RDMA-memcached: 14.76us
Datasets: Uniform data sets, same value size (32B)

Different Workloads

Throughput: RFP 5.5 MOPS for all workloads

Shared data structure in RDMA-Memcached
Datasets: Uniform datasets, different data sizes
Workload: 95% Get
Throughput: \( \text{RFP} = 2.6 \sim 3.8 \times (\text{ServerReply or RDMA-Memcached}) \) (value size 32B ~ 2048B)
Compare with Server-Reply

Different processing time

For getting the value of $R$

**Dataset**: Uniform Dataset with value size 32B

**Workloads**: 95% GET

$R=5$ : Increasing the process time, system should switch to server-reply
Compare with **Server-Reply**

**Different Fetching Size**

For getting the value of $F$

**Dataset**: Uniform Dataset with different value sizes

**Workloads**: 95% GET

256, 512, 640 are all OK for support different applications
Limitations

- Synchronized Communication
  - Extremely low latency requirement
  - Batching?
- Small Size Data Communication
  - Data center applications
- Asymmetry System Configuration
  - MPI or MapReduce?
  - Preferred by key-value stores or databases.
Conclusion

• Based on two observations:
  ➢ Performance asymmetry of In-bound and Out-bound operations
  ➢ Access Amplification in server bypass

• New paradigm RFP: support RPC, with high performance using server in-bound RDMA operations. The evaluation results shows the benefits.
Final Remark 1: behind asymmetry.

Q: What on earth makes the asymmetry, hardware? software?
A: Please look at the asymmetry data path. Out-bound RDMA is issued by Sender’s CPU. No receiver CPU is used. This is asymmetry. Higher performance InfiniBand devices with lower performance CPU will make more asymmetry.
Final Remark 2: Multi-processes

Usually one RDMA context will be used for each process. The context might be a performance limitation. Multiple processes can use more contexts, thus improve the performance.

4 machines (1 server with 3 clients). (Intel E5-2407 2.4Ghz x 4, ConnectX-4 x 1) per machine
Thank You!!
Q&A
Compare with *Server-Reply*

**Dataset**: Skew Dataset
**Workloads**: different workloads
**Throughput**: RFP 5.5 still MOPS (not influenced)
Compare with Server-Reply

Dataset: Skew Dataset, Workloads: 95% GET

- **RFP** performs **best** in average latency
- **ServerReply** is still limited by the RNIC’s **out-bound** RDMA-write
- **RDMA-Memcached** is bounded by the **CPU** at the server side
The number of retries in RFP under different workloads

<table>
<thead>
<tr>
<th></th>
<th>Uniform</th>
<th></th>
<th>Skewed</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>95% GET</td>
<td>5% GET</td>
<td>95% GET</td>
<td>5% GET</td>
</tr>
<tr>
<td>Percentage of $N &gt; 1$</td>
<td>0.105%</td>
<td>0.13%</td>
<td>0.09%</td>
<td>0.09%</td>
</tr>
<tr>
<td>The largest $N$</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>

This kind of occasional case(*the number of retries can be as large as 9*) **never** repeatedly appears, so there will not be an unnecessary switch between *RFP* and *server-reply*
Different Queue Pair Types

- **Reliable Connection (RC)**
  All the server-bypass solutions include RFP (The only queue type that supports both one-sided RDMA_READ and RDMA_Write)

- **Unreliable Connection (UC, UD)**
  HERD, Fasst (achieve higher performance)

Techniques such as Doorbell batching can be used for UD-based solution to gain lower latency and higher throughout.

Different Paradigms

- **Server-reply**
  Hbase with RDMA, RDMA-memcached and DARE

- **Server-bypass**
  DrTM, C-Hint and FaRM

- **A combination of server-reply and server-bypass**
  Pilaf and Cell

Pilaf, C-Hint, and FaRM, all of them using server-reply to serve PUT requests cause these systems suffering from the limited performance of server’s out-bound RDMA.
## Basic APIs in RFP

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>client send(server id, local buf, size)</code></td>
<td>Client sends message (kept in local buf) to server’s memory through RDMA-write</td>
</tr>
<tr>
<td><code>client recv(server id, local buf)</code></td>
<td>Client remotely fetches message from server’s memory into local buf through RDMA-read</td>
</tr>
<tr>
<td><code>server send(client id, local buf, size)</code></td>
<td>Server puts message for client into local buf</td>
</tr>
<tr>
<td><code>server recv(client id, local buf)</code></td>
<td>Server receives message from local buf</td>
</tr>
<tr>
<td><code>malloc buf(size)</code></td>
<td>Allocate local buffers that are registered in the RNIC for message transferring through RDMA</td>
</tr>
<tr>
<td><code>free buf(local buf)</code></td>
<td>Free local buf that is allocated with malloc buf</td>
</tr>
</tbody>
</table>
The server thread number is 6 and the value size is 32 bytes. The workload is uniform and read-intensive (95% GET)
RFP automatically switches to the server-reply mode for reducing clients’ CPU utilization when the request process time becomes longer.
Compare with **Server-Reply**

**Different Fetching Size**

Increasing the fetching just lowers the performance of RFP in average due to network resource wasting.

**Skewed Workload**

the peak throughput of RFP is still **5.5** MOPS under 5%, 50%, and 95% GET percentages.
Overall Performance Metrics

\[ T = \sum_{i=1}^{M} T_i, \text{ where } T_i = \begin{cases} I_{R,F} & F \geq S_i \\ I_{R,F}/2 & F < S_i \end{cases} \]

\[ T = \arg\max_{R,F} f(R, F, P, S) \]

- **T** – System Throughput
- **R** – the retrying number of RDMA Read from clients before it switches to server-reply mode;
- **F** – the fetching size used by the clients to read remote results from server;
- **P** – the process time for requests on server;
- **S** – the RPC call result sizes.

- P and S related to applications only.
- R and F are related to both applications and the RDMA hardware.