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

Maomeng Su[1], Mingxing Zhang[1], Kang Chen[1],
Zhenyu Guo[2], Yongwei Wu[1]

清華大學 Tsinghua University

Microsoft Research
微软亚洲研究院

[1]Tsinghua University
[2]Microsoft Research

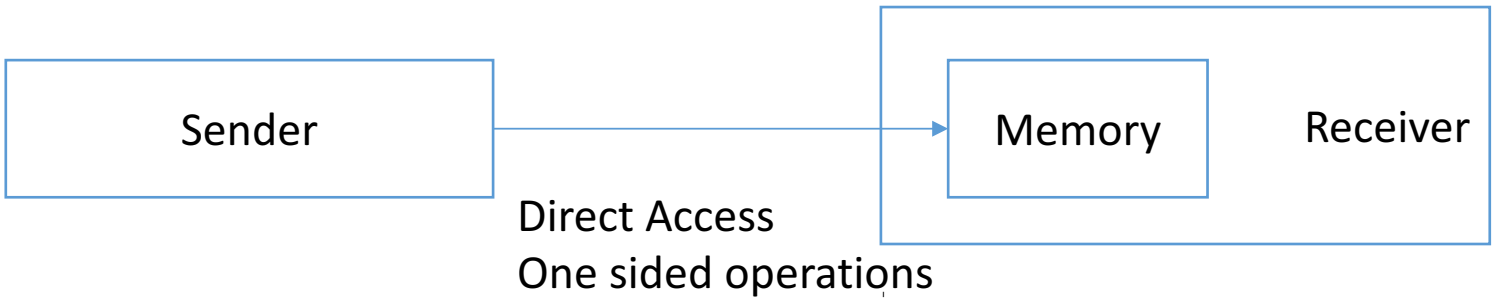
RDMA in modern data centers

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**

  Programmability is good

- **Server-Bypass**

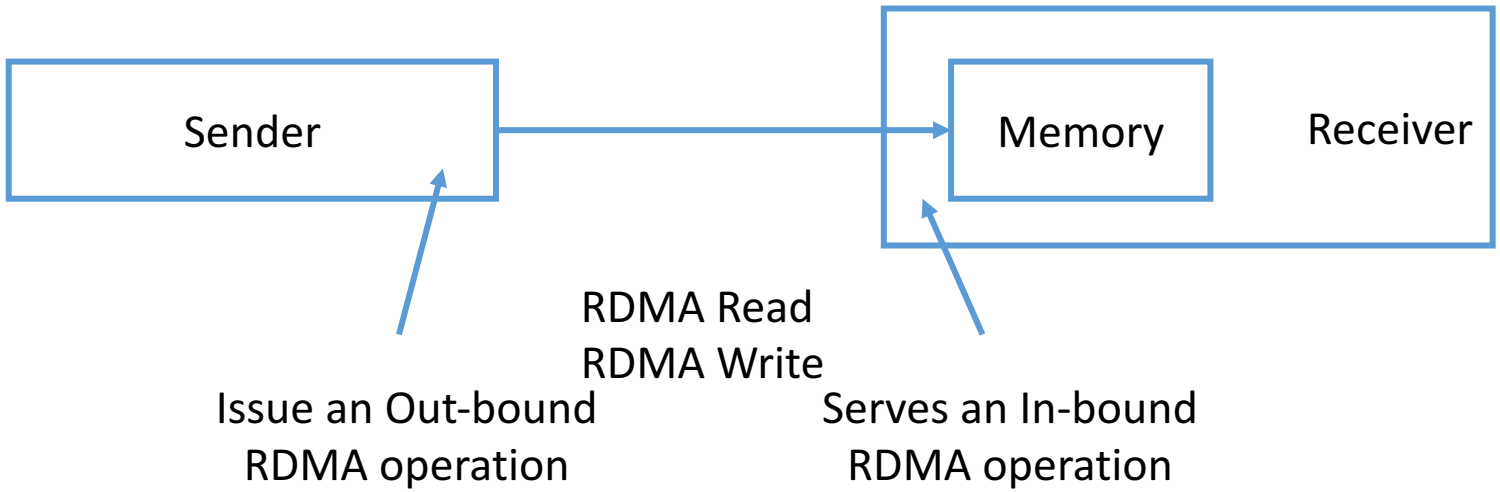
  Pilaf [ATC 2013]
  FaRM-KV [NSDI 2014]
  C-Hint [SOCC 2014]

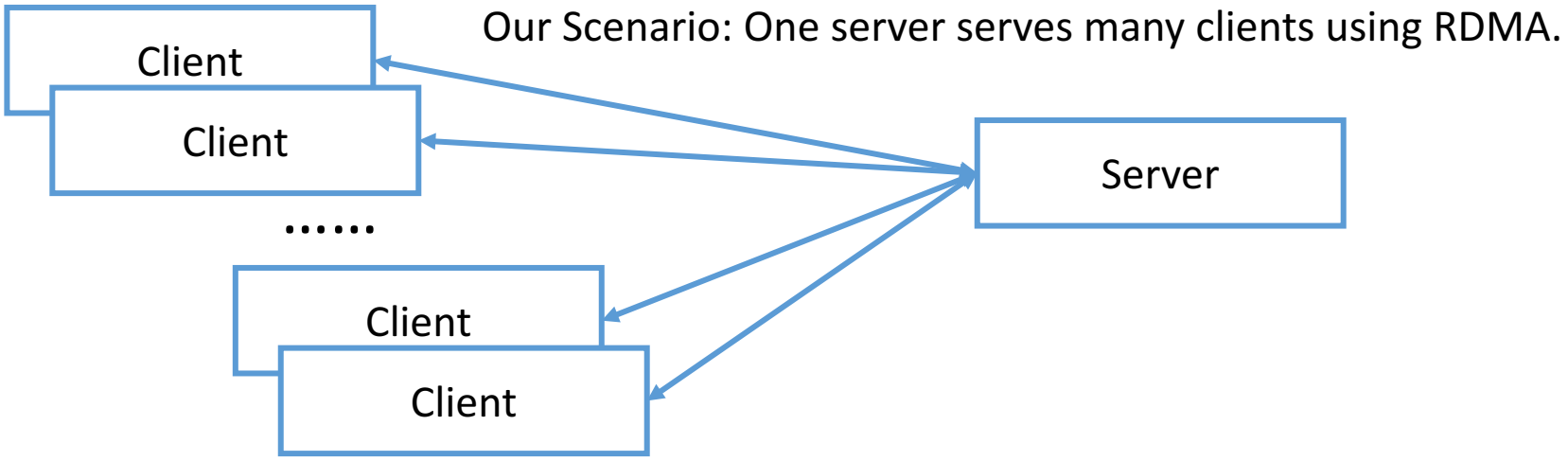
- **Features**

  Clients access remote data structures by RDMA_Read/Write
  Totally bypass Server CPU

  Need great efforts

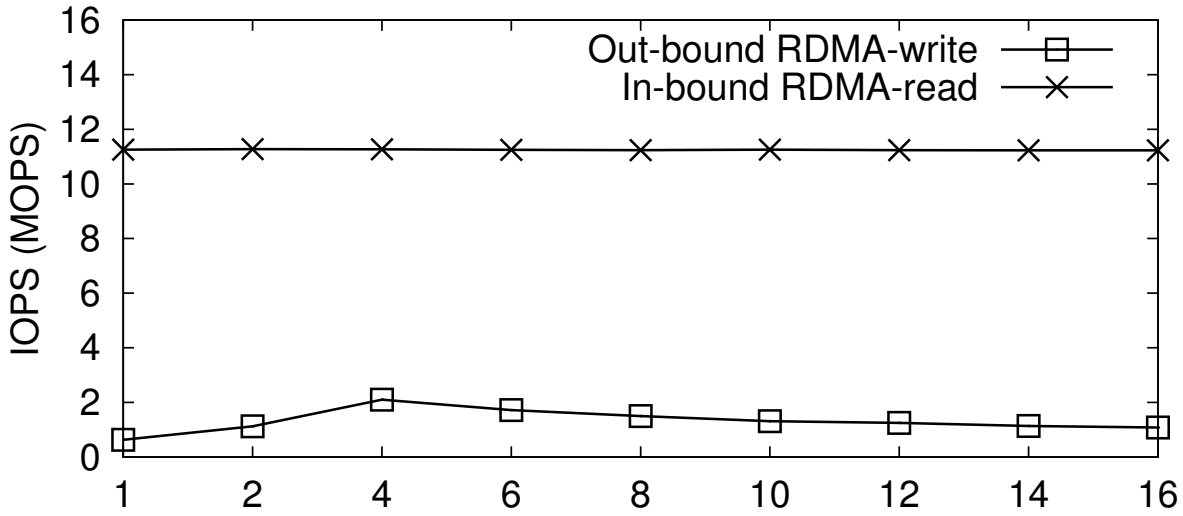
In-bound RDMA, Out-bound RDMA

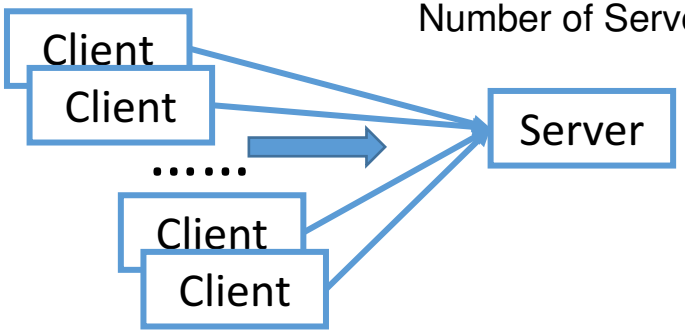




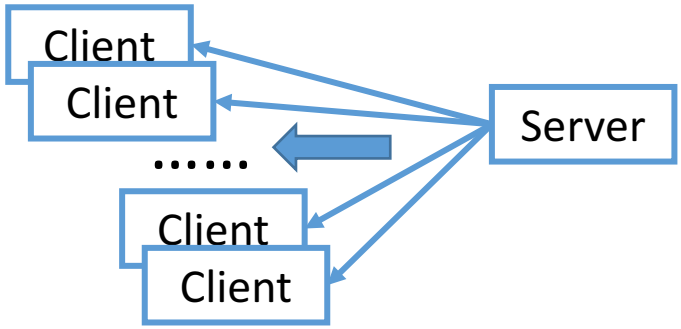
In-bound vs. Out-bound Asymmetry

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





In-bound Testing: RDMA Read



Out-bound Testing: RDMA Write

Observation1

- **In-bound vs. Out-bound Asymmetry**

*** Overhead: Issuing RDMA > Serving RDMA**

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

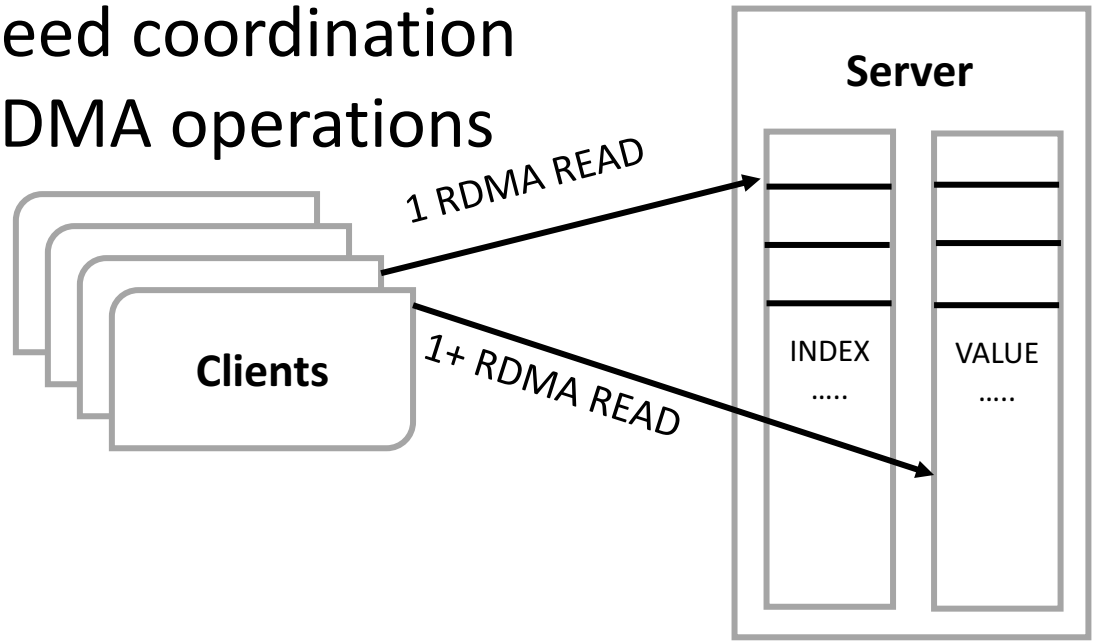
Limitation of server-reply mode

Bypass Access Amplification

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

What about programmability?


```
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;
}
```

Special detection

Special data structures

```
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;
}
```

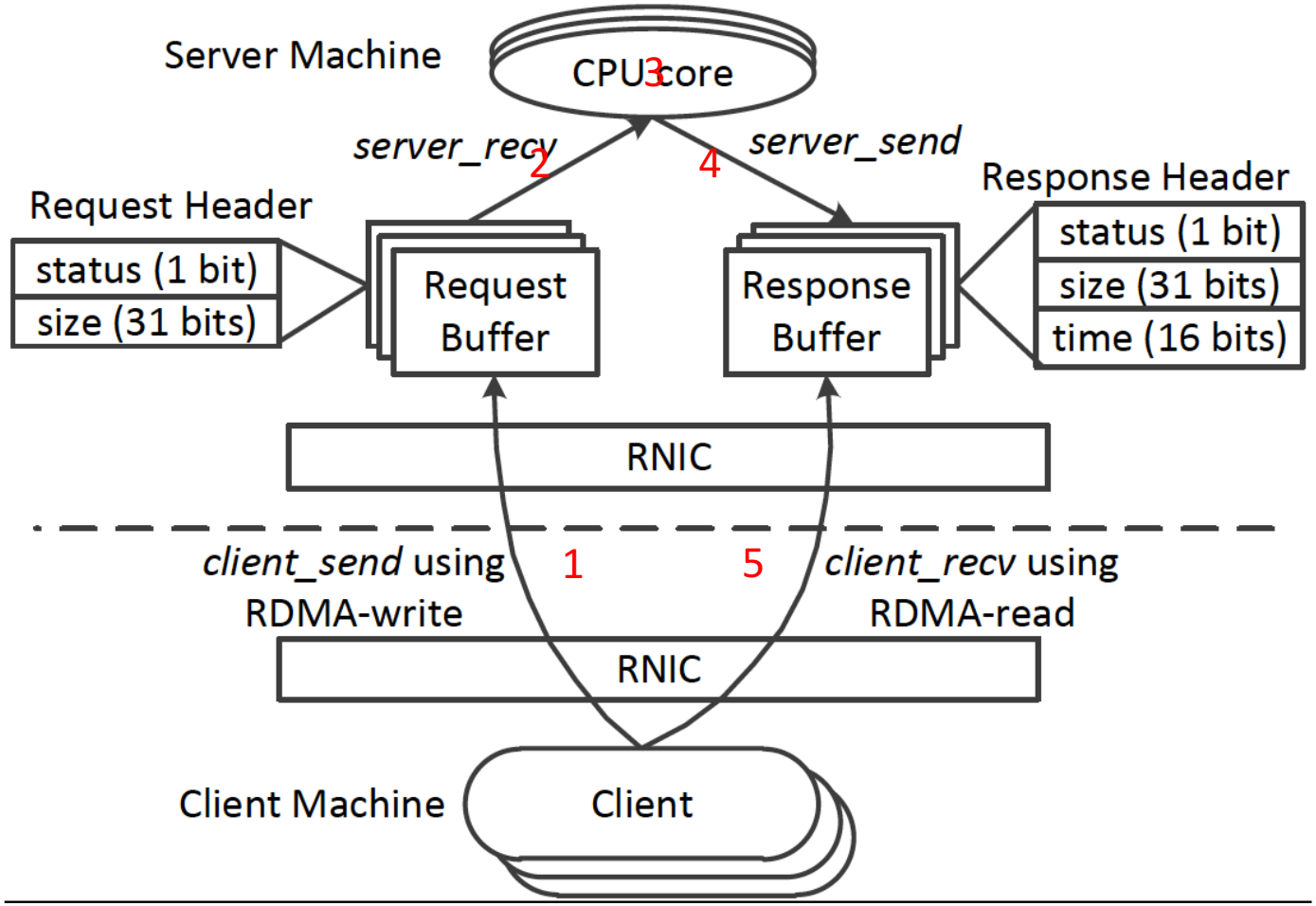
Familiar with server-reply mode.

Design Choices for RPC system

Paradigm	Programmability	Performance Limitation
Server-Reply	Good	Limited by out-bound performance
Server-Bypass	Need great effort, Special data structures	Limited by number of retries

RPC Phases	Request Send	Request Process	Result Return
Server-reply	In-bound RDMA	Server involved	Out-bound RDMA
Server-bypass	In-bound RDMA	Server bypassed	In-bound RDMA
RFP	In-bound RDMA	Server involved	In-bound RDMA
Meaningless	In-bound RDMA	Server bypassed	Out-bound RDMA

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.

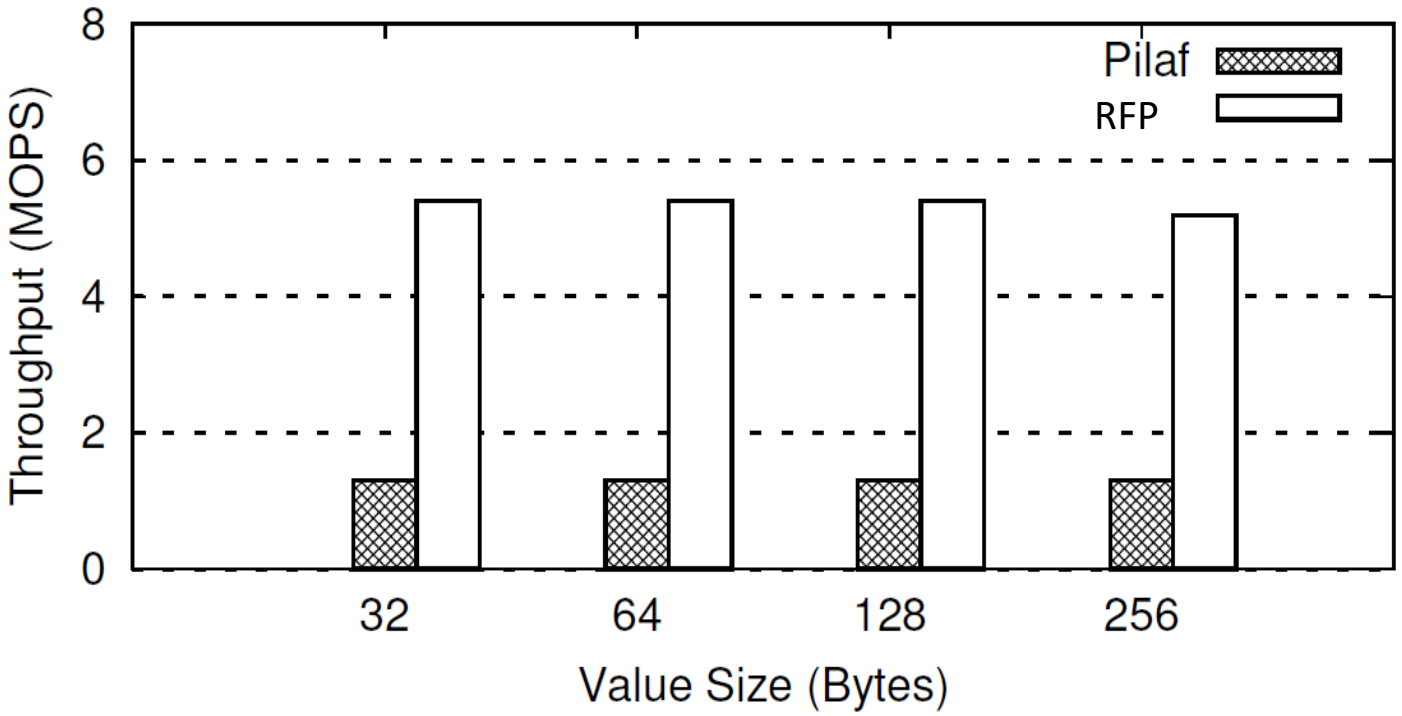
Evaluation

- ✓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

- **Setup**
  - 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 distribition with parameter .99), generated by YCSB. 128 million keys (key size 8-byte).
  - **Workloads:** Different GET percentiles (95%, 50%, 5%)

Compare with *Server-Bypass*



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

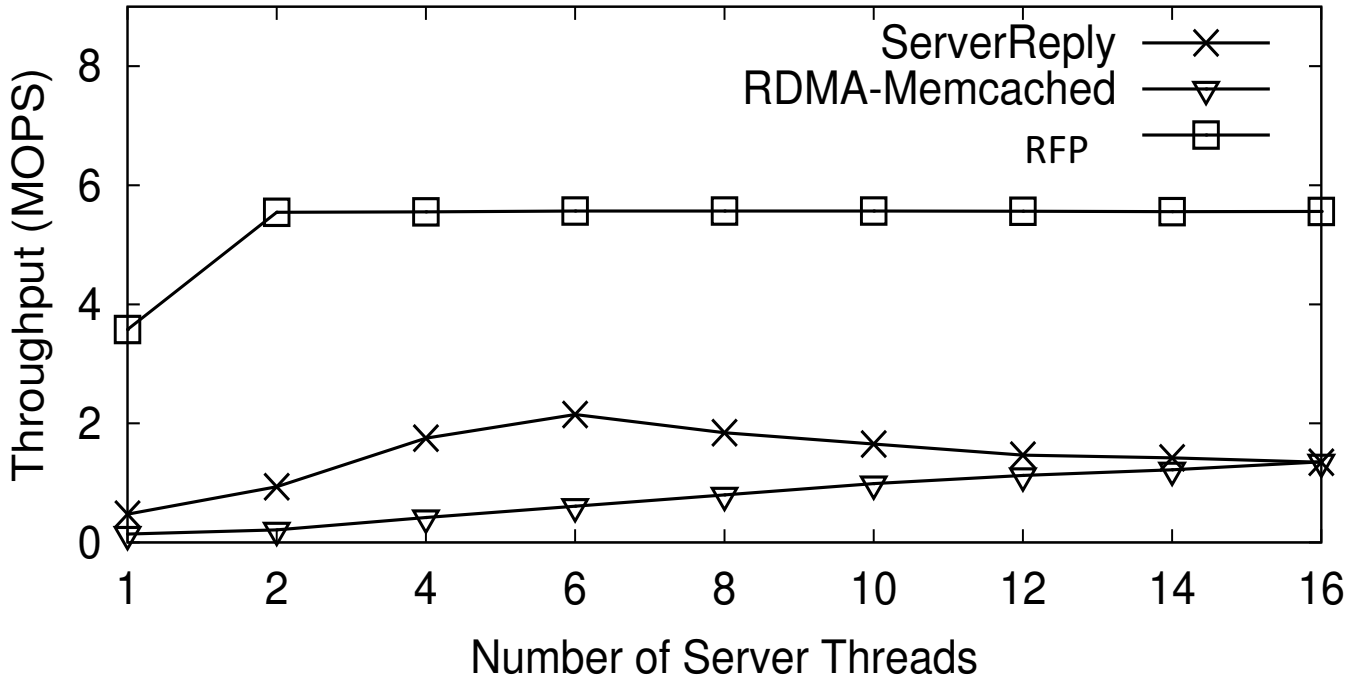
Compare with Server-Reply

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)

Compare with *Server-Reply*

Throughput on uniform dataset with same value size



Datasets: Uniform dataset, same value size (32B)

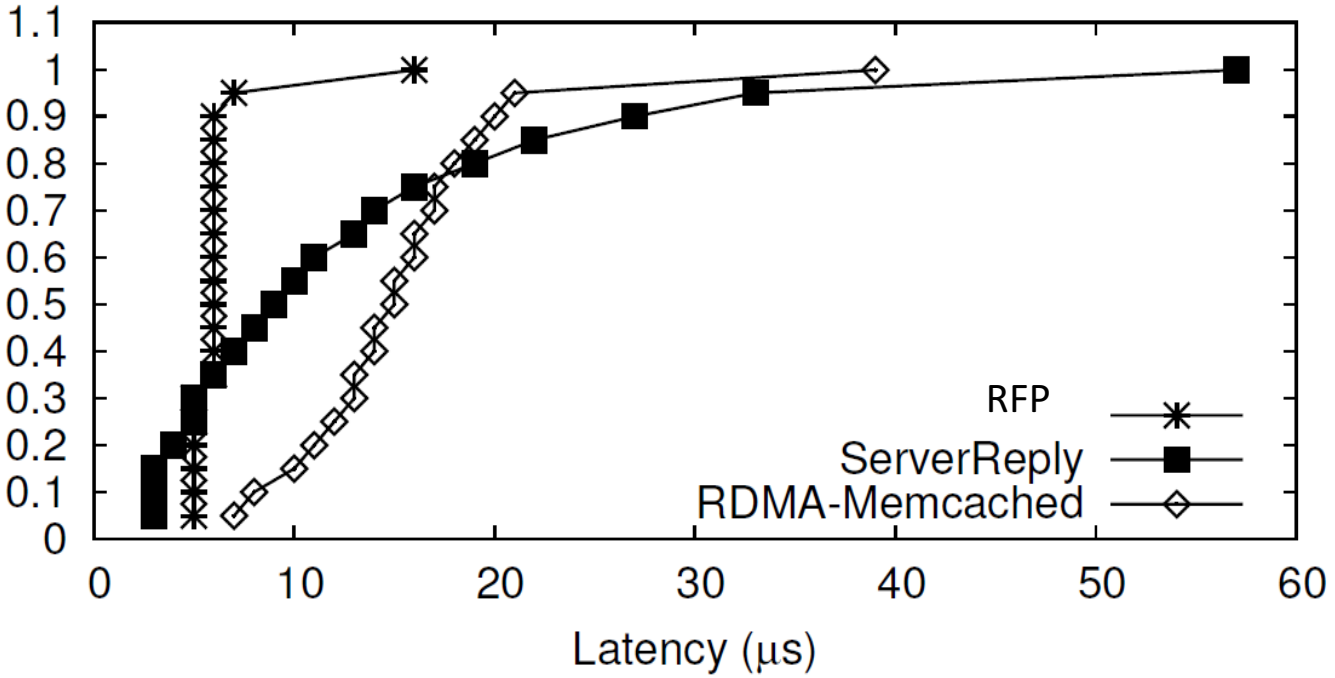
Workloads: 50% GET

Throughput: RFP = 2.6 x ServerReply

= 4.1 x RDMA-Memcached

Compare with *Server-Reply*

Latency on uniform dataset with same value size



Datasets: Uniform dataset, same value size (32B)

Workloads: 50% GET

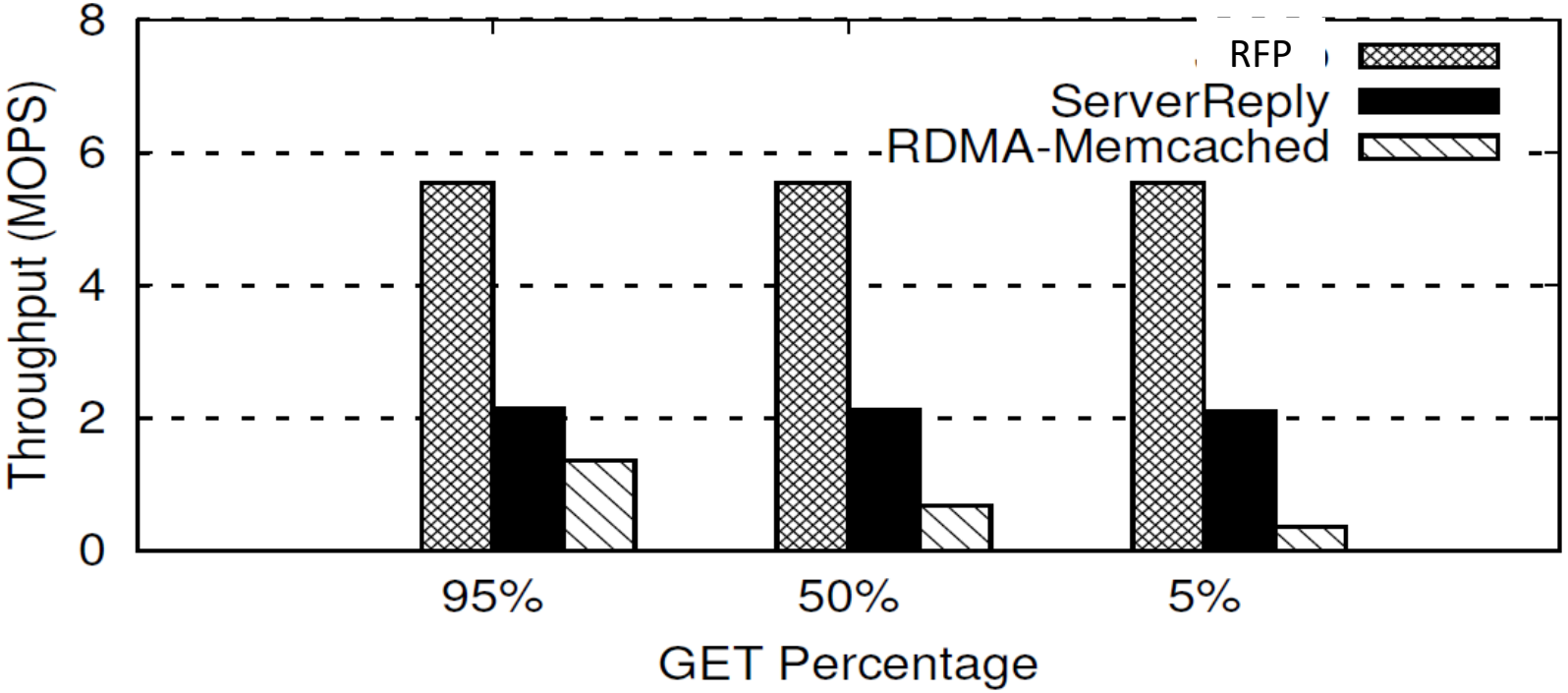
Latency: RFP, 5.78 us

ServerReply: 12.06us,

RDMA-memcached: 14.76us

Compare with *Server-Reply*

Throughput on uniform dataset with different workloads



Datasets: Uniform data sets, same value size (32B)

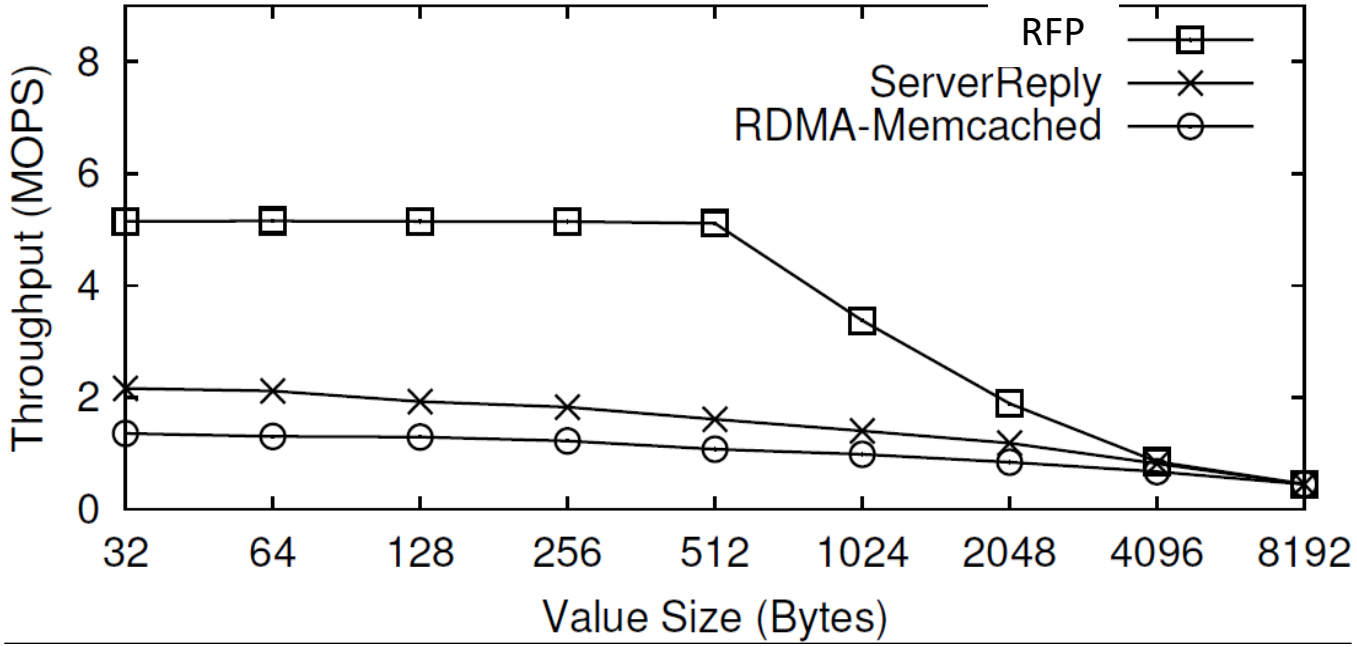
Different Workloads

Throughput: RFP 5.5 MOPS for all workloads

Shared data structure in RDMA-Memcached

Compare with *Server-Reply*

Throughput on uniform dataset with different data sizes



Datasets: Uniform datasets, different data sizes

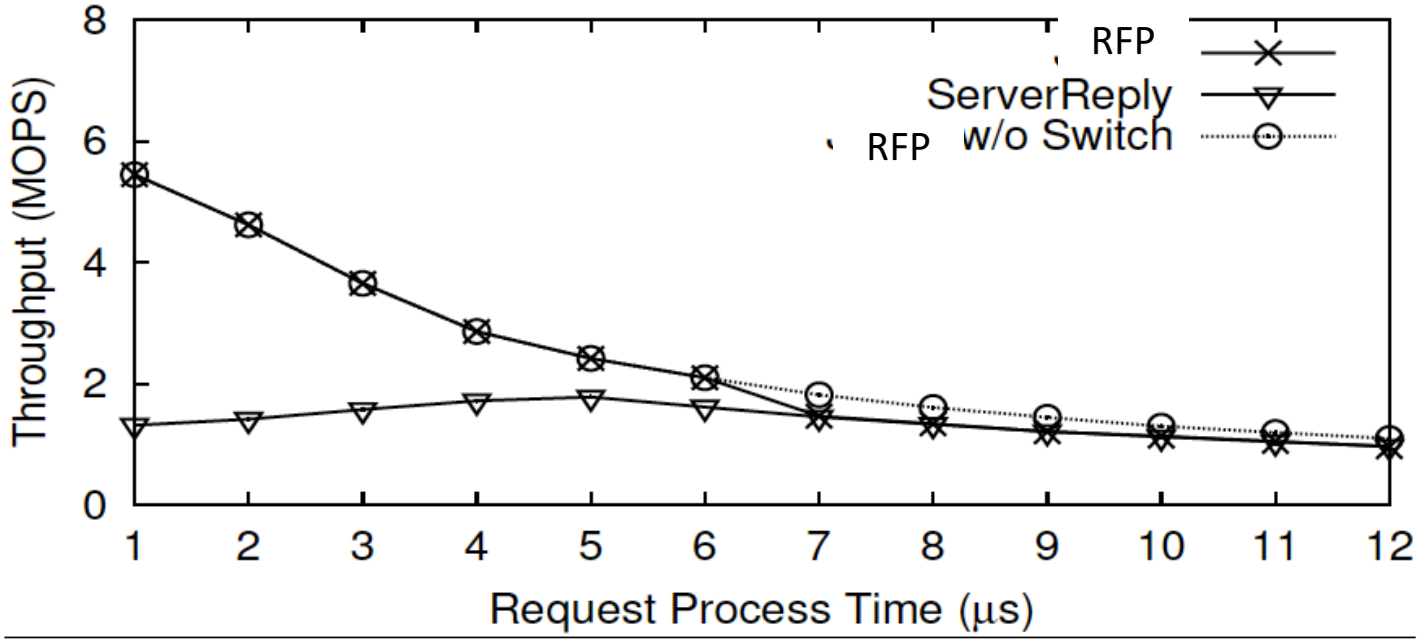
Workload: 95% Get

Throughput: RFP = 2.6~3.8 x (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

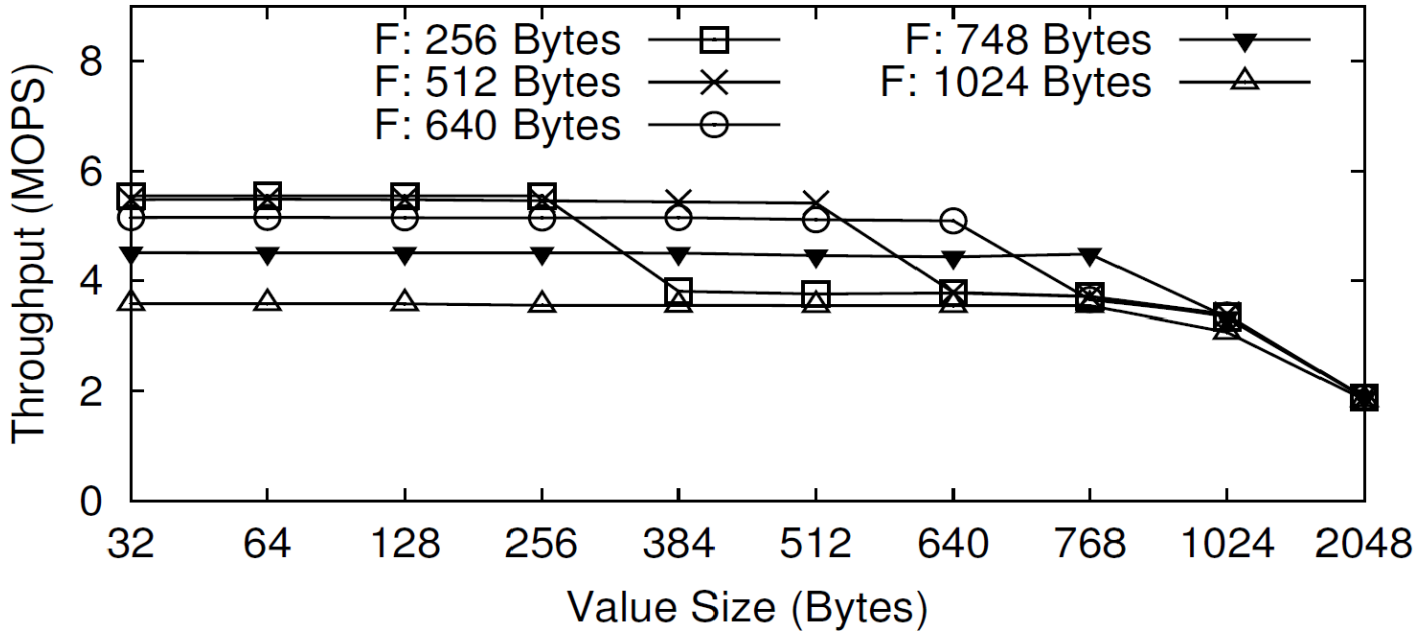
Workloads: 95% GET

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

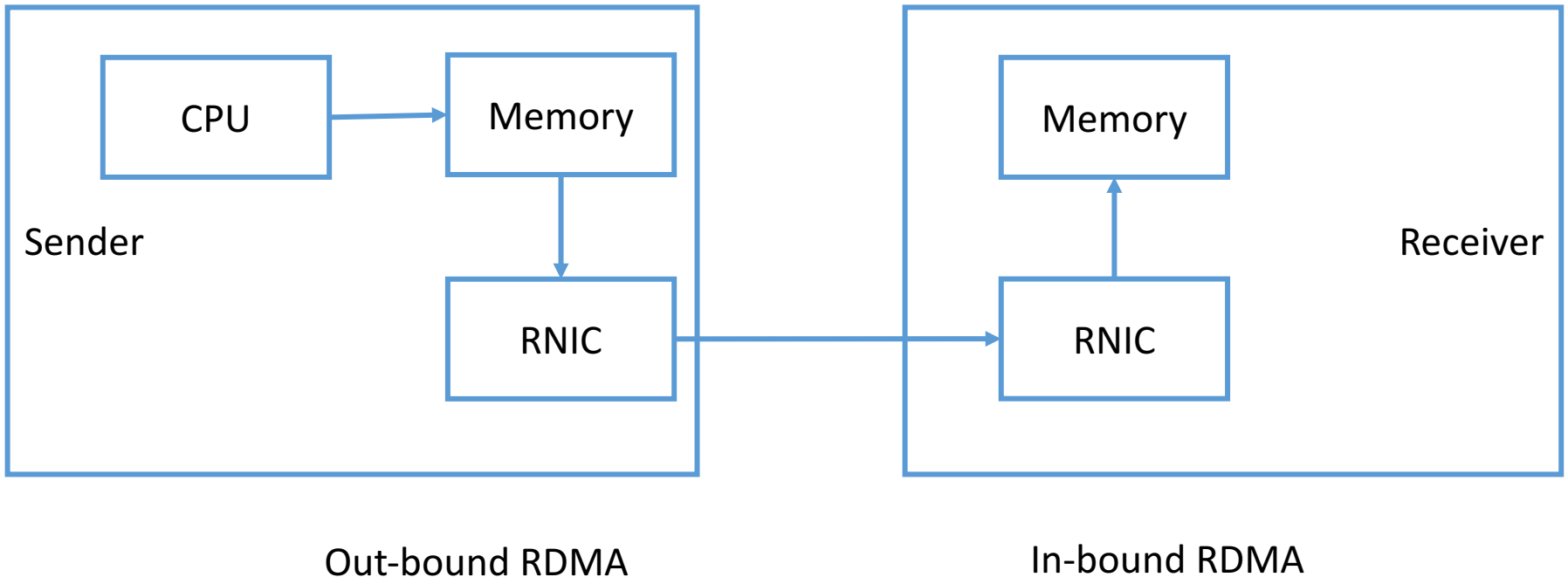
- Synchronized Communication
  - Extremely low latency requirement
  - Batching?
- Small Size Data Communication
  - Data center applications
- Asymmetry System Configuration
  - MPI or MapReduce?
  - Preferred by key-value stores or databases.

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.

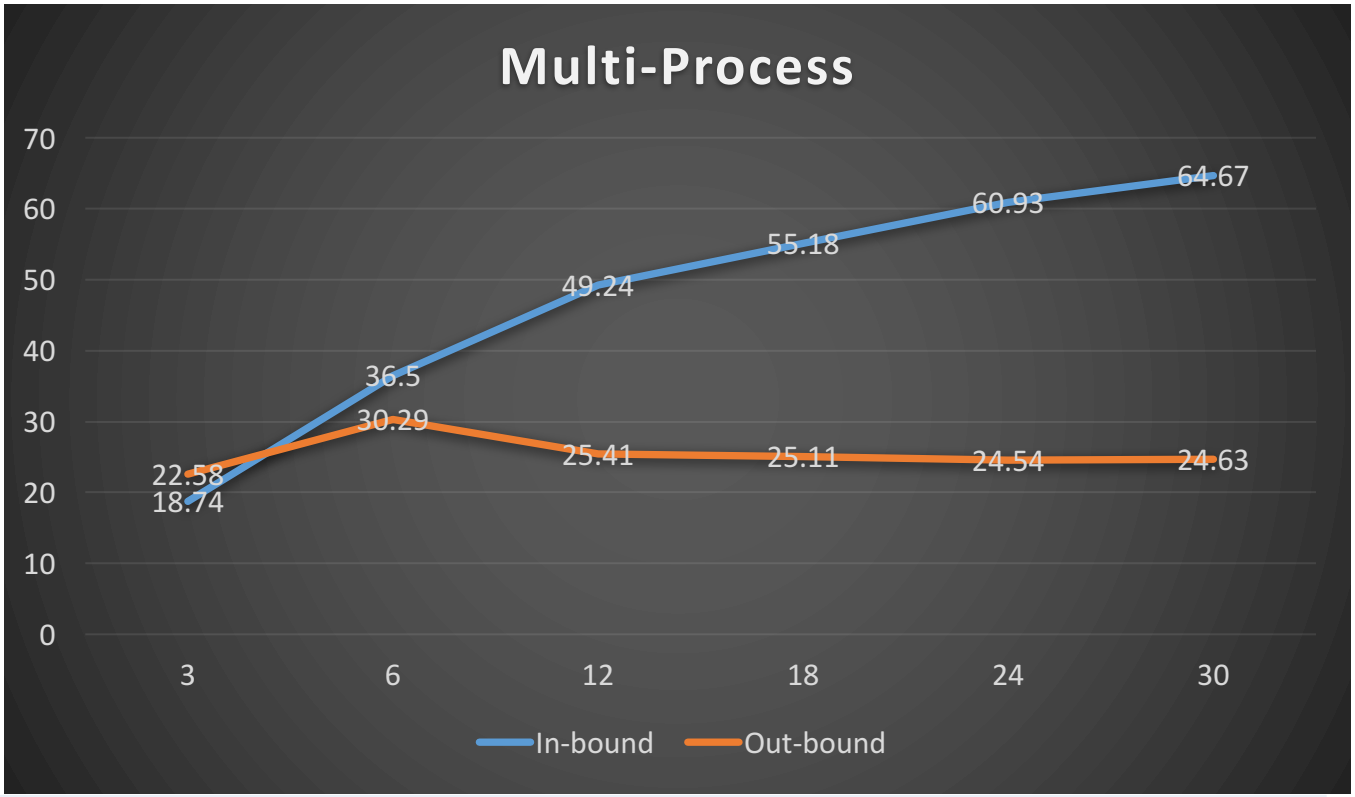
Q: What on earth makes the asymmetry, hardware? software?
A: Please look at the asymmetry data path. Out-bound RDMA is issued by Sender's CPU. No receiver CPU is used. This is asymmetry. Higher performance InfiniBand devices with lower performance CPU will make more asymmetry.



Final Remark 2: Multi-processes

Usually one RDMA context will be use for each process. The context might be a performance limitation.

Multiple processes can use more contexts, thus improve the performance

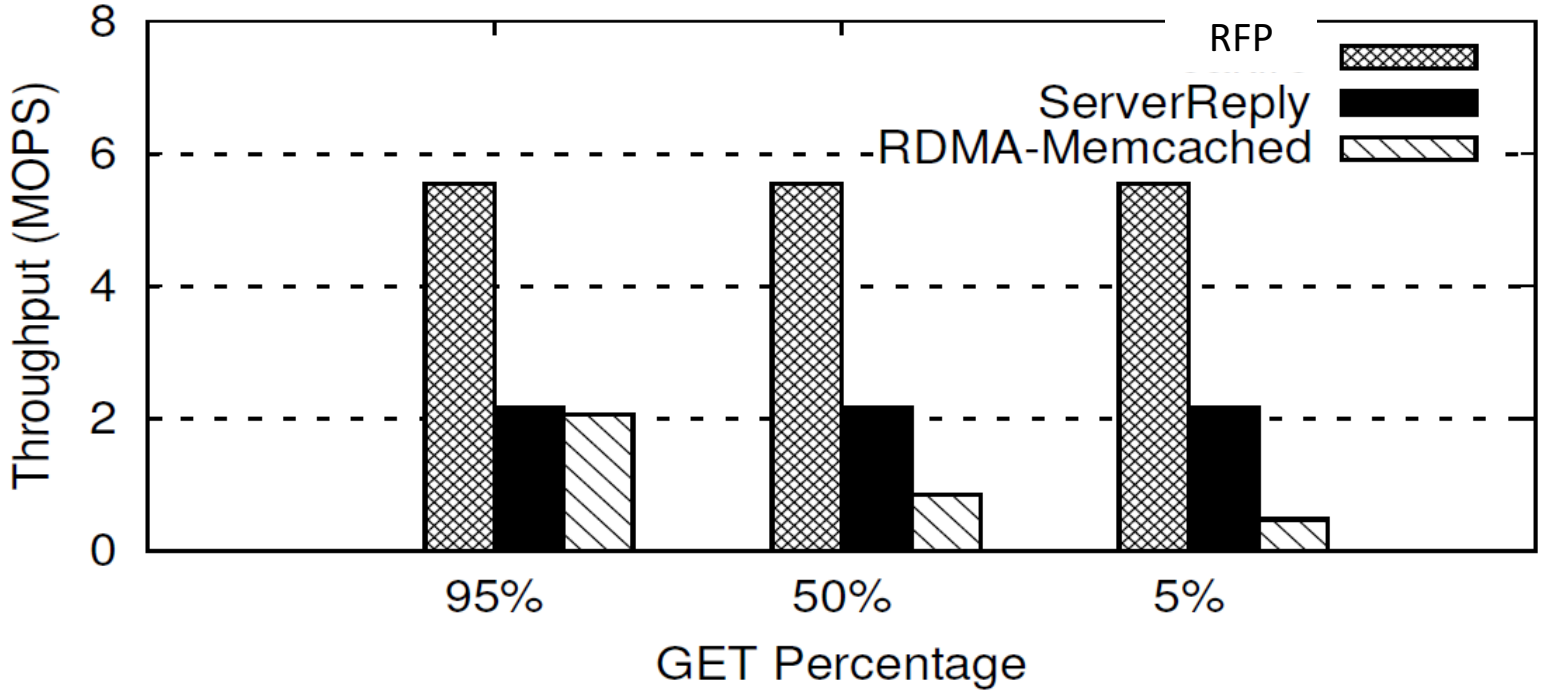


4 machines (1 server with 3 clients). (Intel E5-2407 2.4Ghz x 4, ConnectX-4 x 1) per machine

Thank You!!
Q&A

Compare with *Server-Reply*

Throughput on skew dataset with different workloads



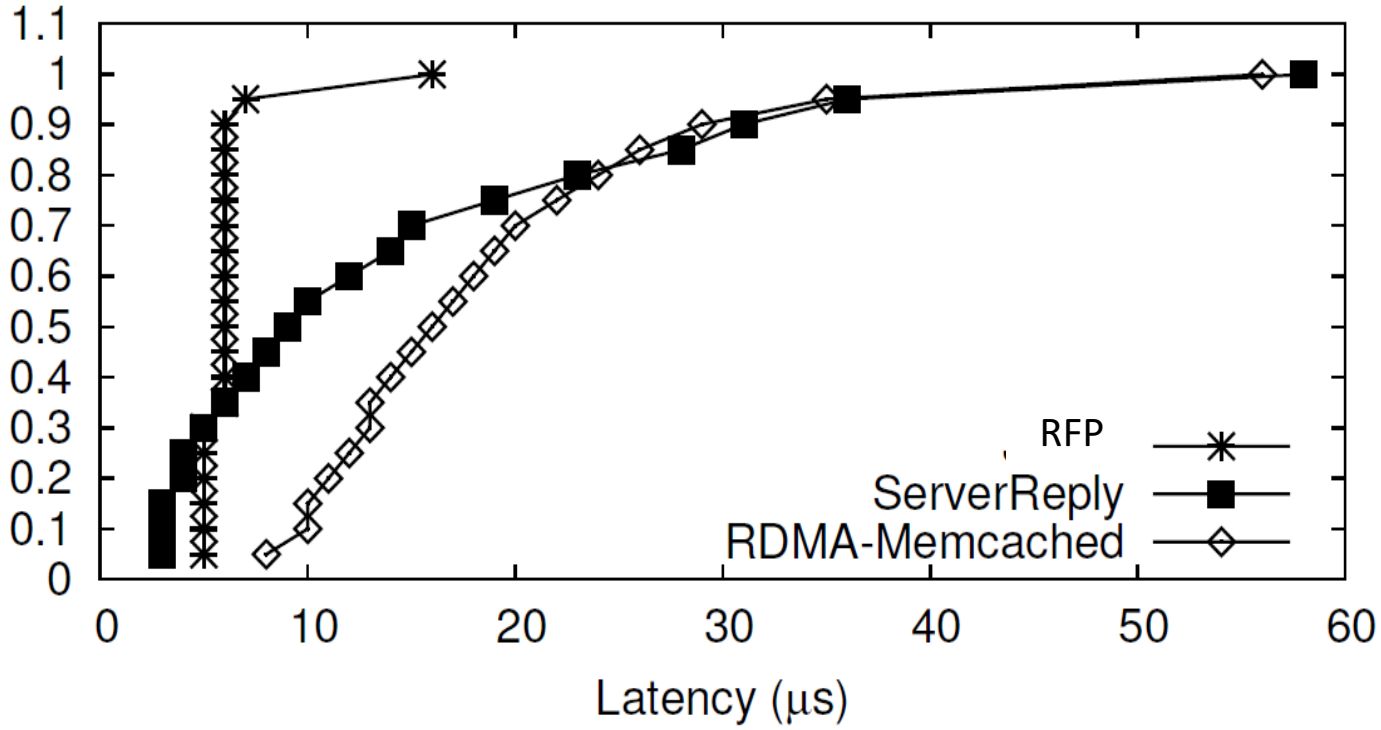
Dataset: Skew Dataset

Workloads: different workloads

Throughput: RFP 5.5 still MOPS (not influenced)

Compare with *Server-Reply*

Latency on skew dataset



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

Compare with *Server-Reply*

The number of retries in RFP under different workloads

	Uniform		Skewed	
	95% GET	5% GET	95% GET	5% GET
Percentage of N > 1	0.105%	0.13%	0.09%	0.09%
The largest N	6	5	9	4

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*

Related Work

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*

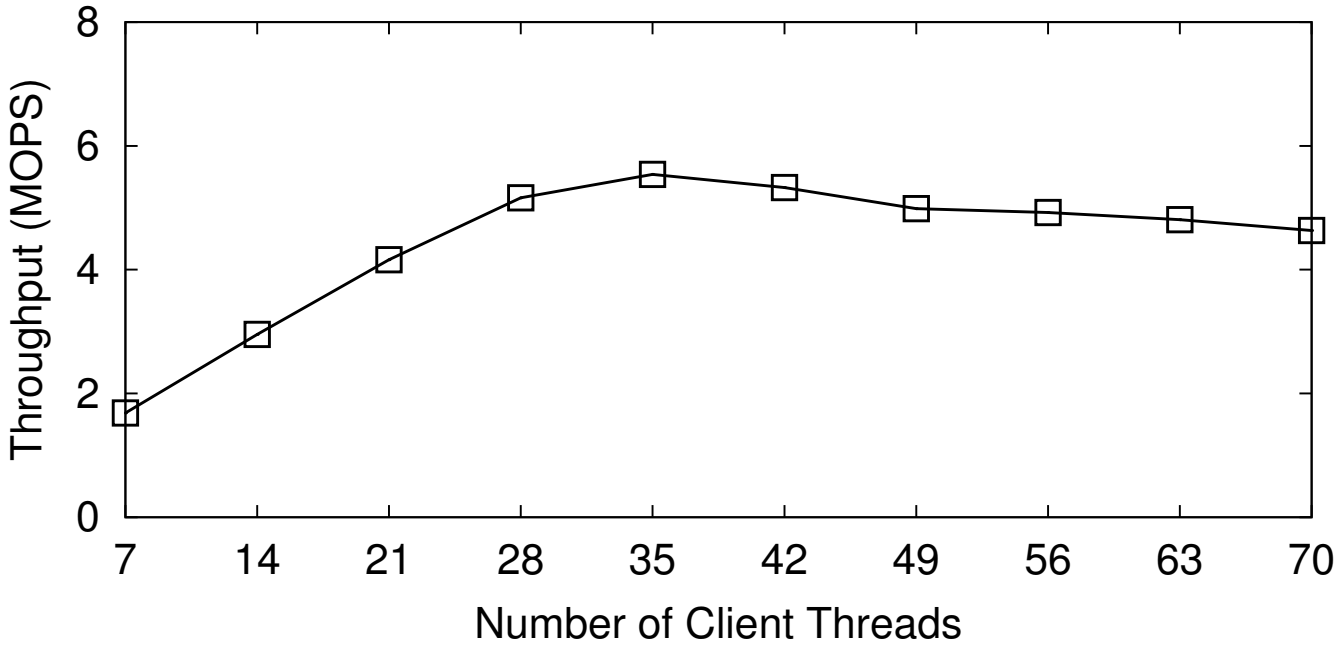
Pilaf, C-Hint, and FaRM, all of them using server-reply to serve PUT requests cause these systems suffering from the limited performance of server's out-bound RDMA.

Backup1 RFP(Remote Fetching Paradigm) Overview

Basic APIs in RFP

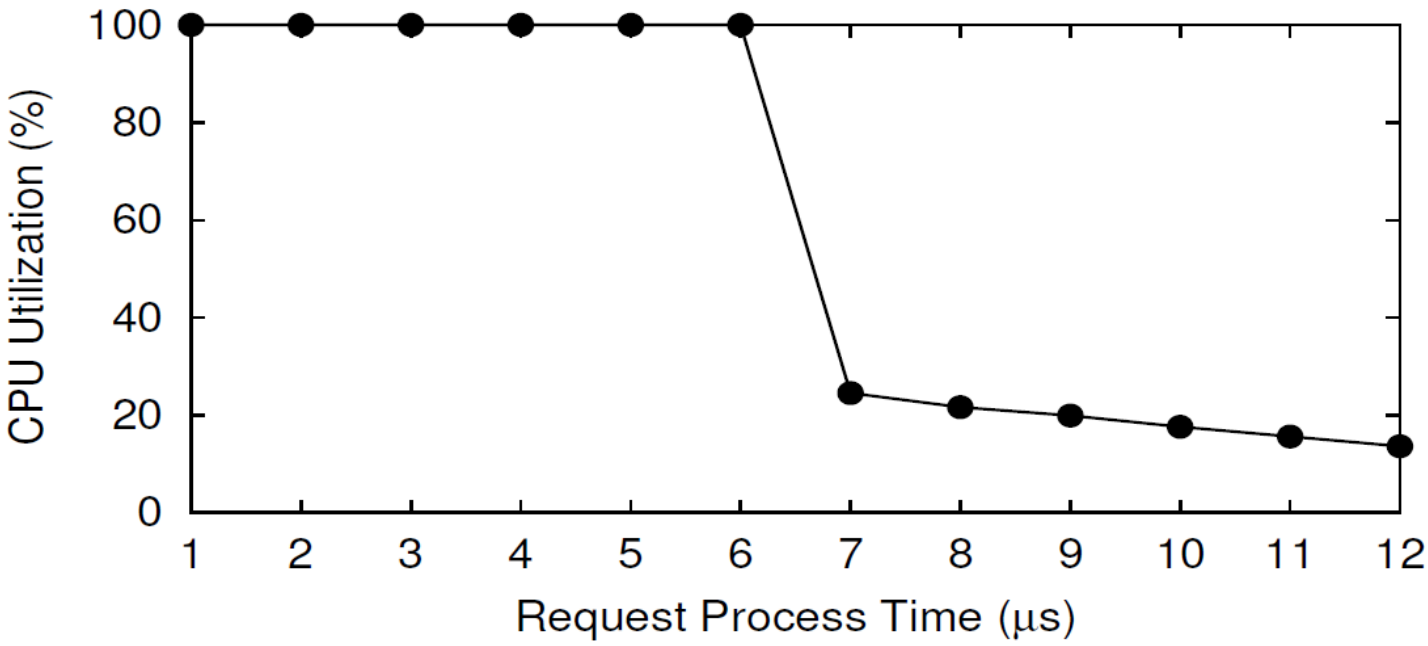
	Request Send
client send(server id, local buf, size)	Client sends message (kept in local buf) to server's memory through RDMA-write
client recv(server id, local buf)	Client remotely fetches message from server's memory into local buf through RDMA-read
server send(client id, local buf, size)	Server puts message for client into local buf
server recv(client id, local buf)	Server receives message from local buf
malloc buf(size)	Allocate local buffers that are registered in the RNIC for message transferring through RDMA
free buf(local buf)	Free local buf that is allocated with malloc buf

Performance of RFP



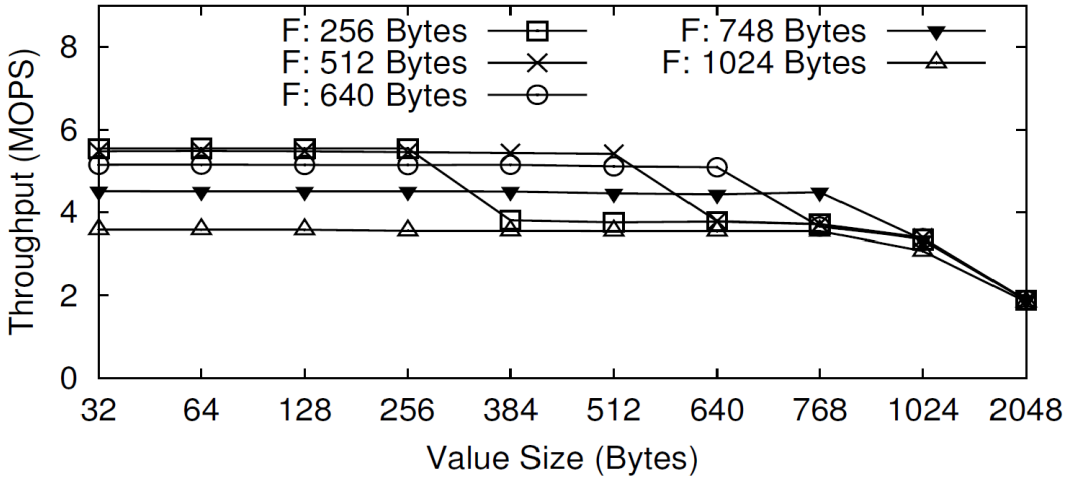
The server thread number is 6 and the value size is 32 bytes. The workload is *uniform* and *read-intensive*(95% GET)

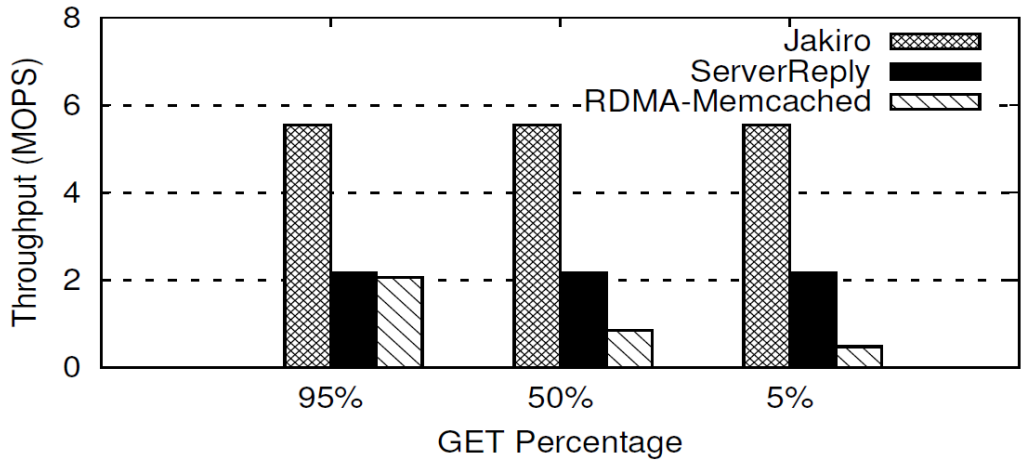
Compare with *Server-Reply*



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

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

$$T = \underset{R,F}{\operatorname{argmax}} f(R, F, P, S)$$

- **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.


idea