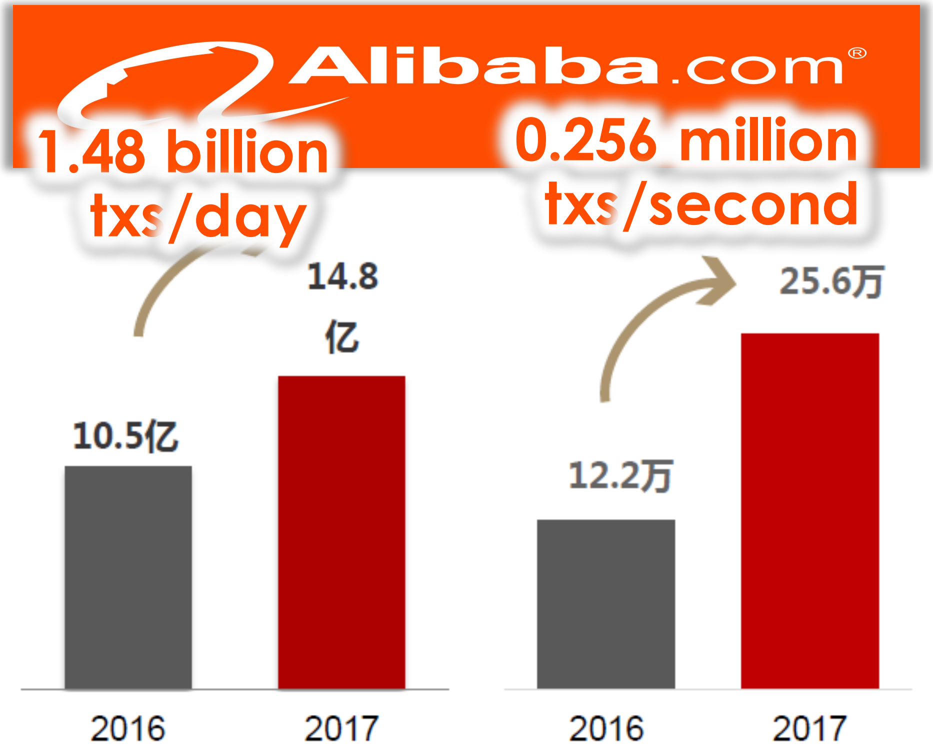
File System Design for Distributed Persistent Memory

Youyou Lu
Tsinghua University

luyouyou@tsinghua.edu.cn
http://storage.cs.tsinghua.edu.cn/~lu

Latency and Throughput Demands



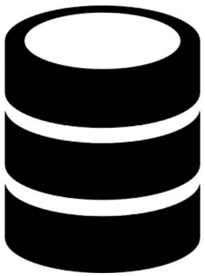
9.56 million
tickets/day

PayPal
11. 6 million
payments/day

In-Memory Storage and Computing

- Data-driven Information Technology
  - Computing-Intensive Computing → Data-Intensive Computing
  - HPC, Big Data, AI

- Low-latency data storage and processing



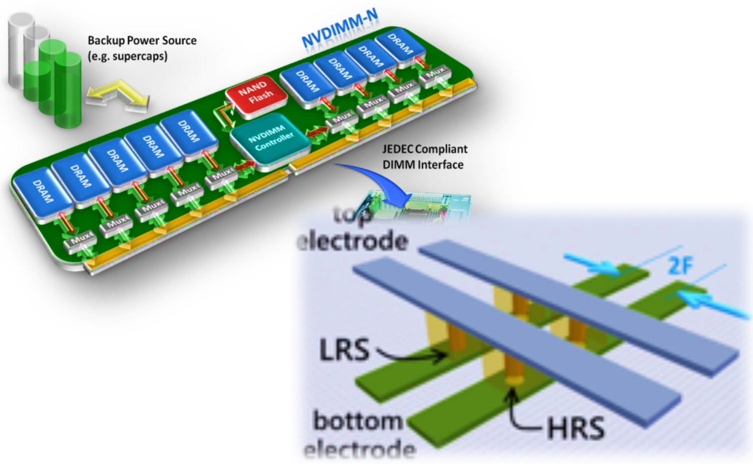
In-memory Databases

In-Memory Data Analytics

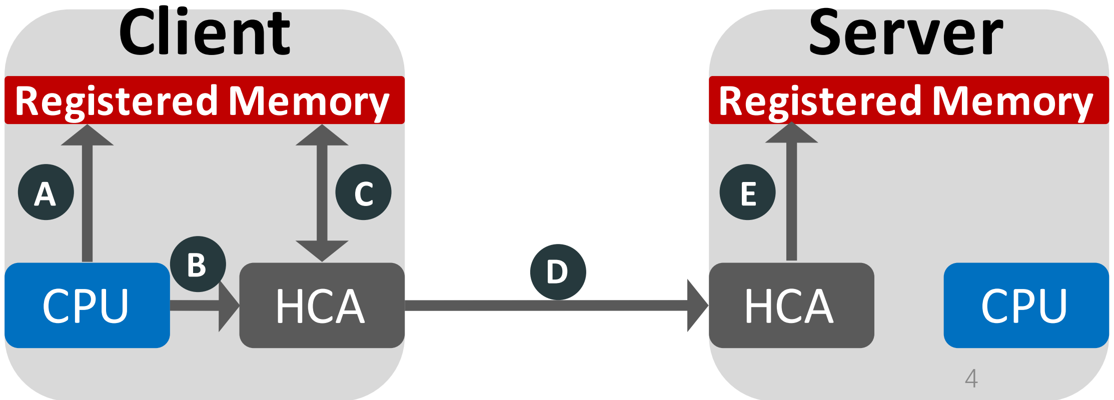
AI System

NVMM & RDMA

- **NVMM** (3D XPoint, etc)
  - Data persistency
  - Byte-addressable
  - Low latency



- **RDMA**
  - Remote direct access
  - Bypass remote kernel
  - Low latency and high throughput

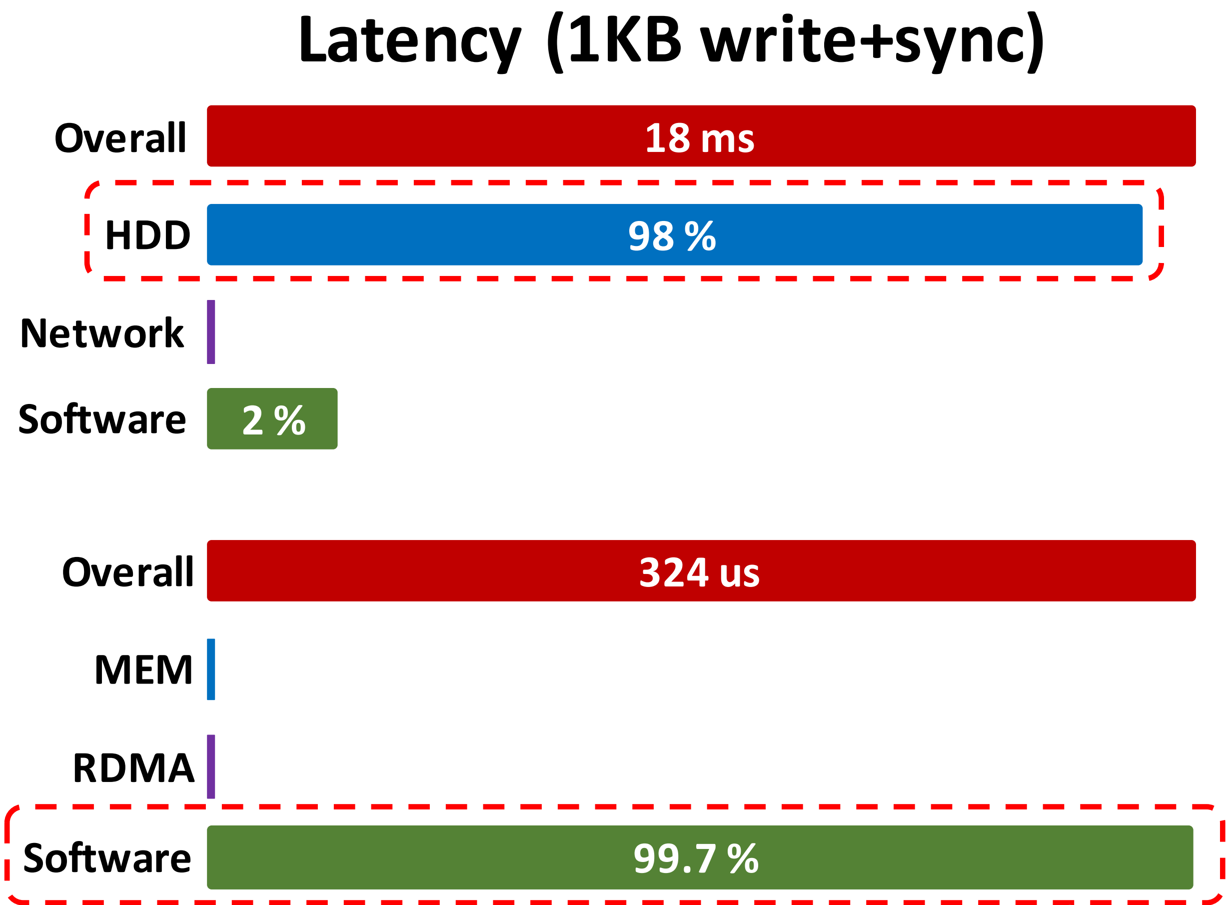


Outline

- Octopus: a RDMA-enabled Distributed Persistent Memory File System
  - Motivation
  - Octopus Design
  - Evaluation
  - Conclusion
- Scalable and Reliable RDMA

Modular-Designed Distributed File System

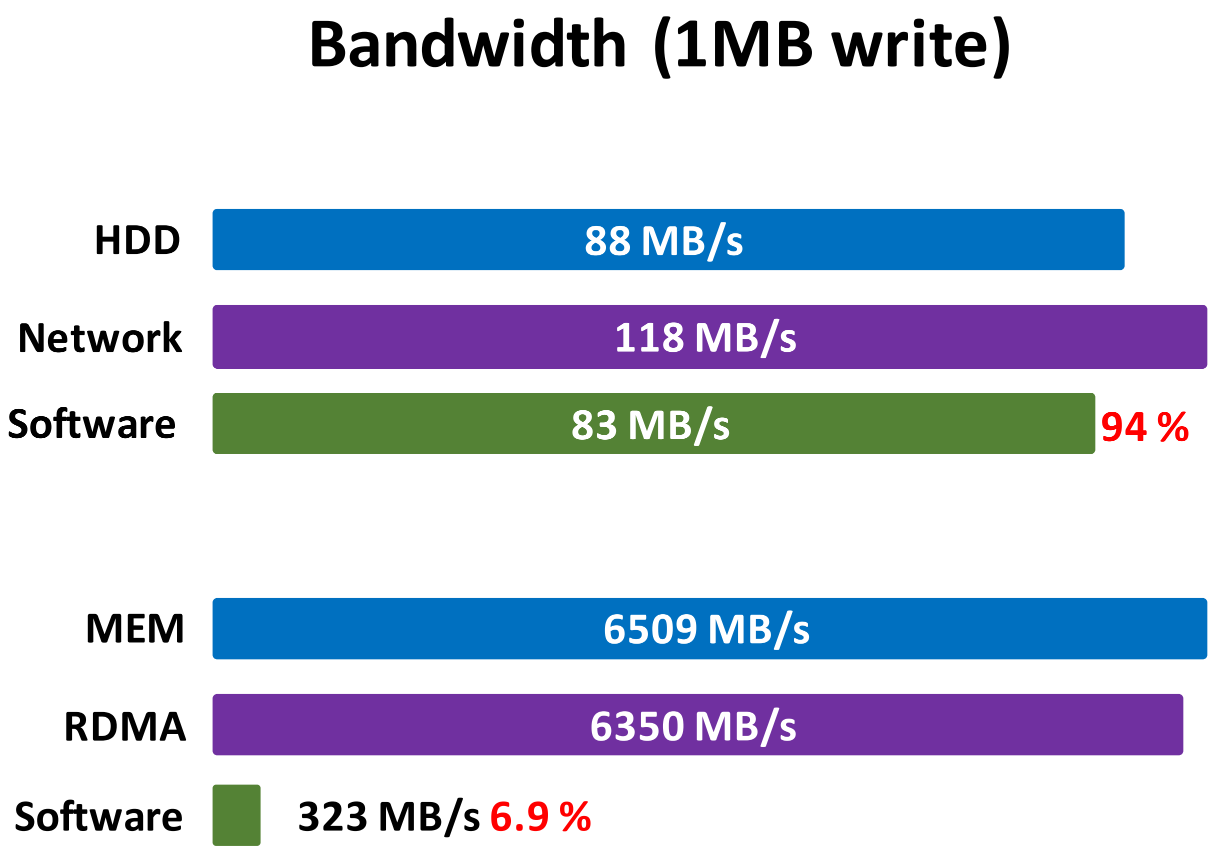
- DiskGluster
  - Disk for data storage
  - GigE for communication
- MemGluster
  - Memory for data storage
  - RDMA for communication



Modular-Designed Distributed File System

- DiskGluster
  - Disk for data storage
  - GigE for communication

- MemGluster
  - Memory for data storage
  - RDMA for communication



RDMA-enabled Distributed File System

- Cannot simply replace the network/storage module
- More than fast hardware …
- New features of NVM
  - Byte-addressability
  - Data persistency
- RDMA verbs
  - Write, Read, Atomics (memory semantics)
  - Send, Recv (channel semantics)
  - Write-with-imm

RDMA-enabled Distributed File System

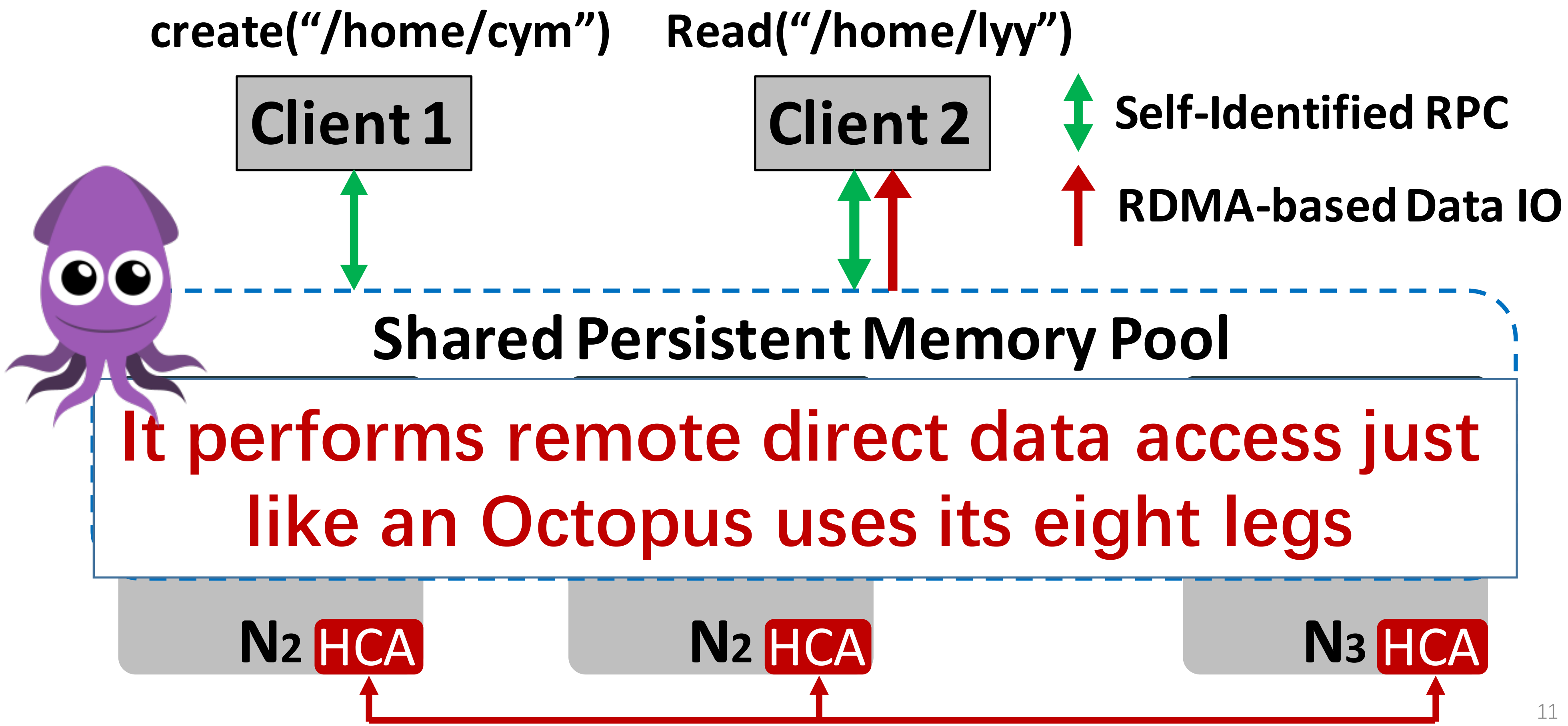
Opportunity		Approaches
Byte-addressability of NVM; One-sided RDMA verbs	→	Shared data managements
One-sided RDMA verbs; CPU is the new bottleneck	→	New data flow strategies
Write-with-Imm	→	Efficient RPC design
RDMA Atomics	→	Concurrent control

- We choose to redesign the DFS!

Outline

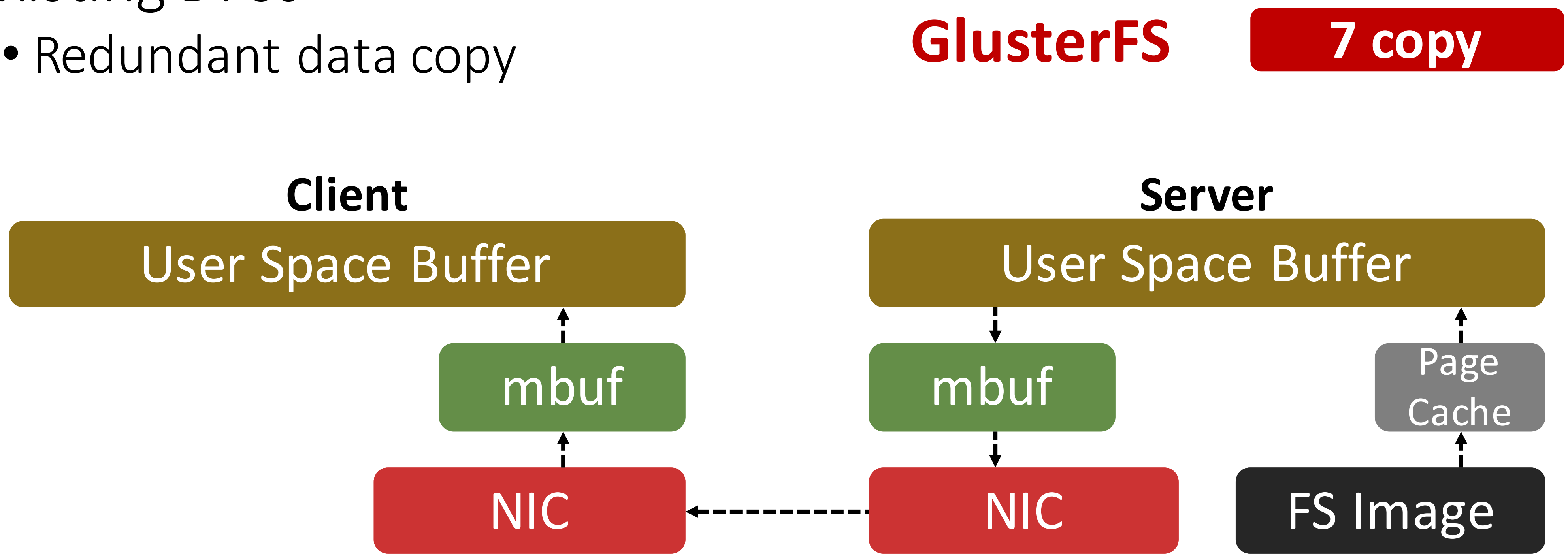
- Octopus: a RDMA-enabled Distributed Persistent Memory File System
  - Motivation
  - Octopus Design
    - High-Throughput Data I/O
    - Low Latency Metadata Access
  - Evaluation
  - Conclusion
- Scalable and Reliable RDMA

Octopus Architecture



1. Shared Persistent Memory Pool

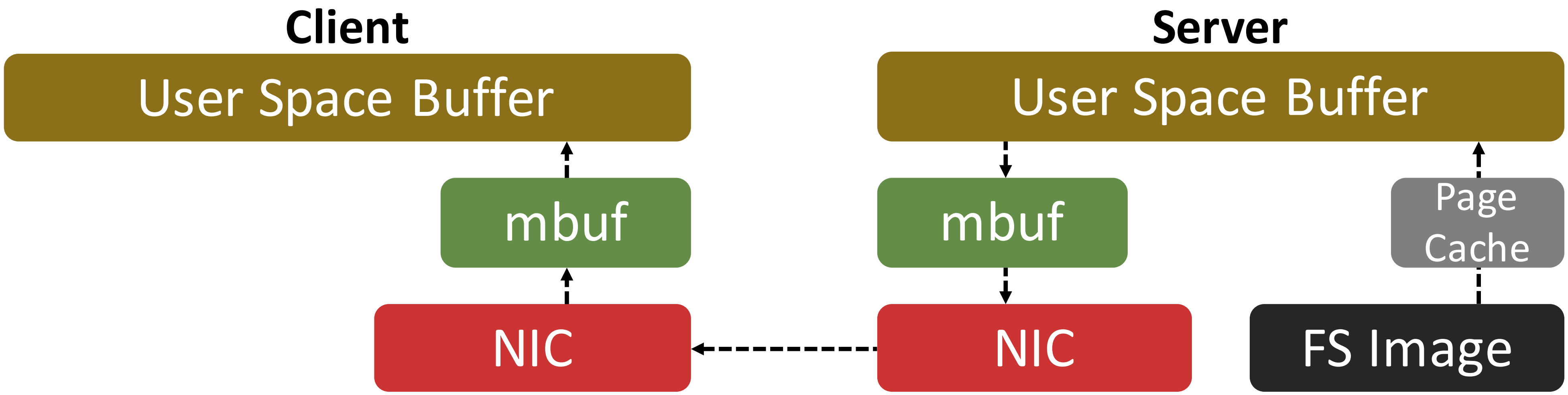
- Existing DFSs
  - Redundant data copy



1. Shared Persistent Memory Pool

- Existing DFSs
  - Redundant data copy

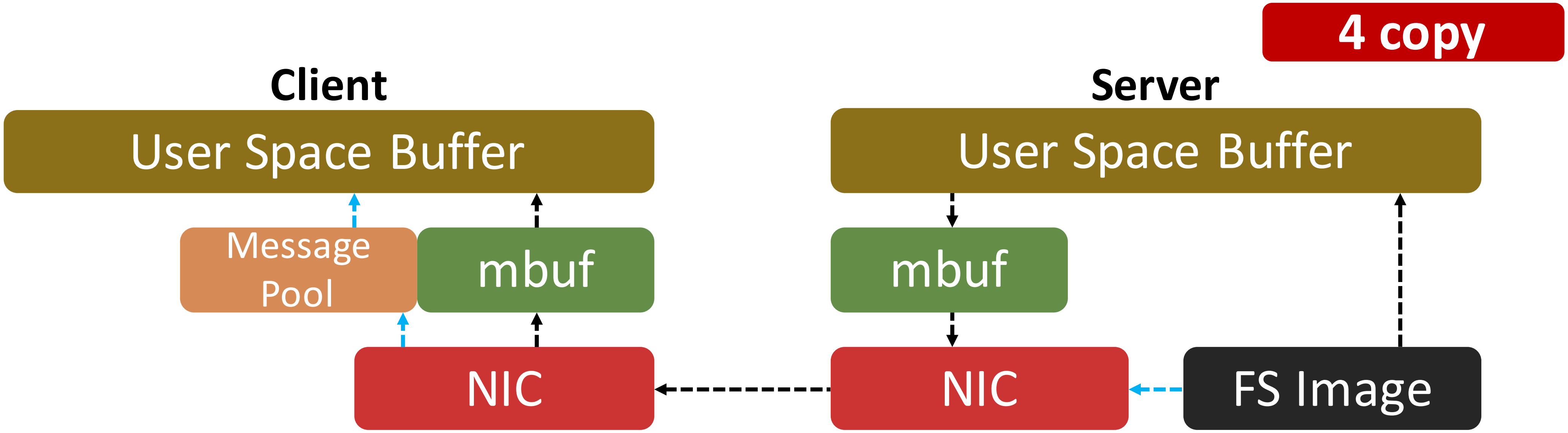
GlusterFS + DAX **6 copy**



1. Shared Persistent Memory Pool

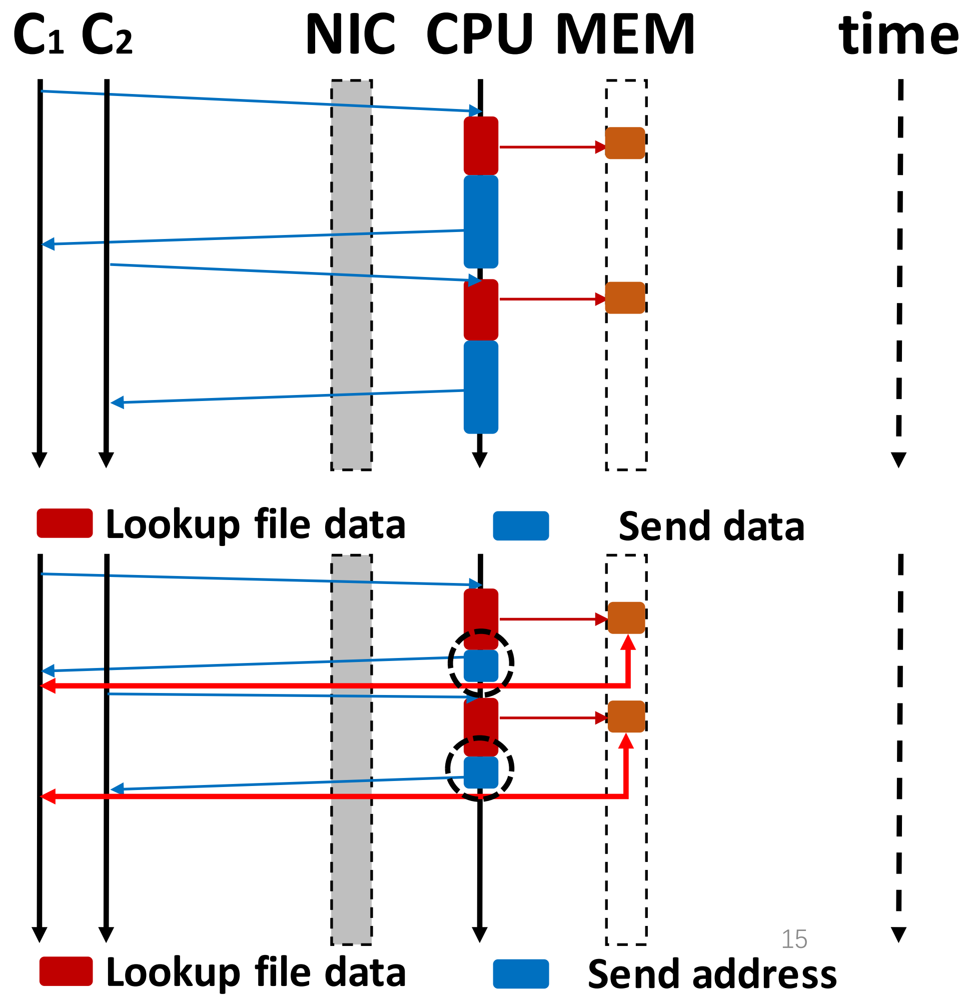
- Existing DFSs
  - Redundant data copy

- **Octopus with SPMP**
  - Introduces the ***shared persistent memory pool***
  - Global view of data layout



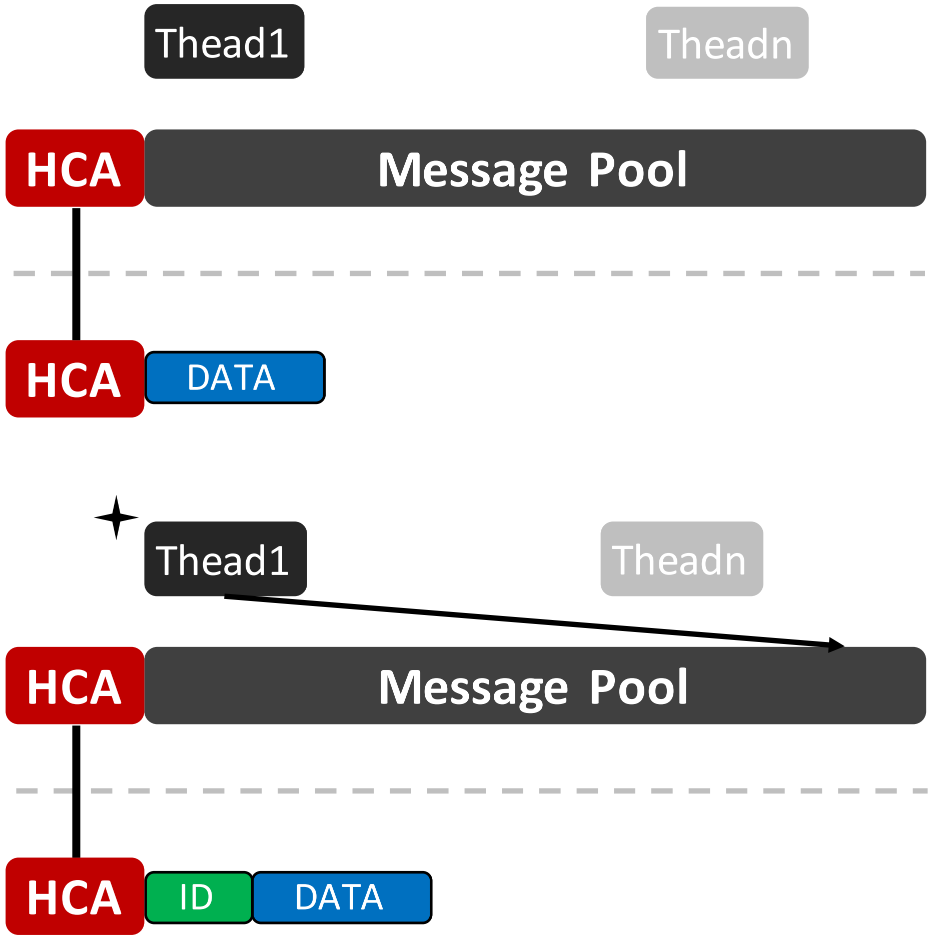
2. Client-Active Data I/O

- Server-Active
  - Server threads process the data I/O
  - Works well for slow Ethernet
  - CPUs can easily become the bottleneck with fast hardware
- Client-Active
  - Let clients read/write data directly from/to the SPMP



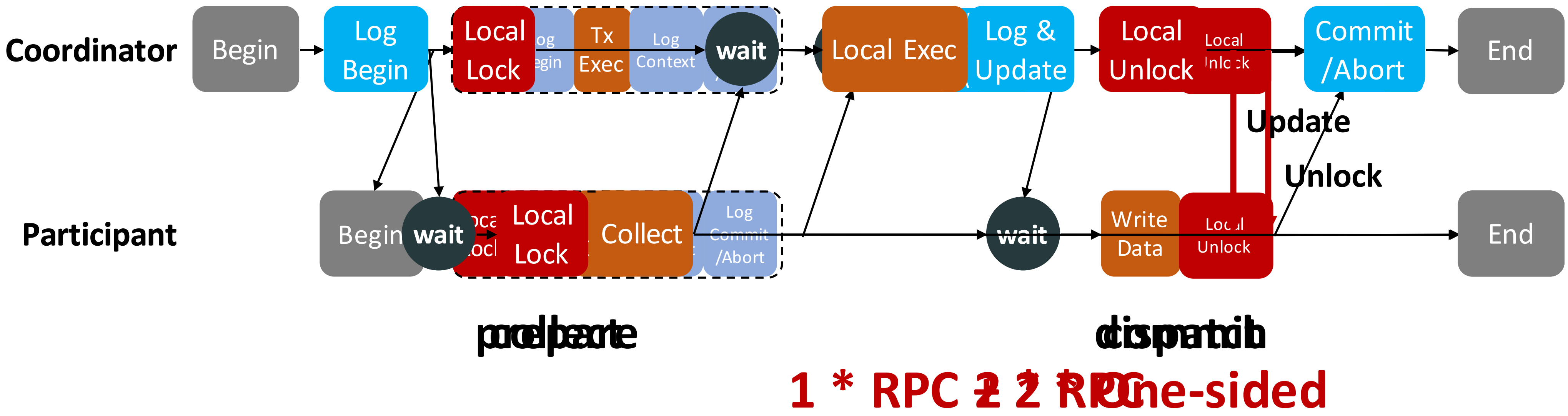
3. Self-Identified Metadata RPC

- Message-based RPC
  - easy to implement, lower throughput
  - **DaRPC, FaSST**
- Memory-based RPC
  - CPU cores scan the message buffer
  - **FaRM**
- Using rdma_write_with_imm?
  - Scan by polling
  - Imm data for self-identification



4. Collect-Dispatch Distributed Transaction

- mkdir, mknod operations need distributed transactions
- Two-Phase Locking & Commit / Collect-Dispatch Transaction
  - Distributed logging / Local logging with remote in-place update
  - Two-phase lock / Distributed lock service



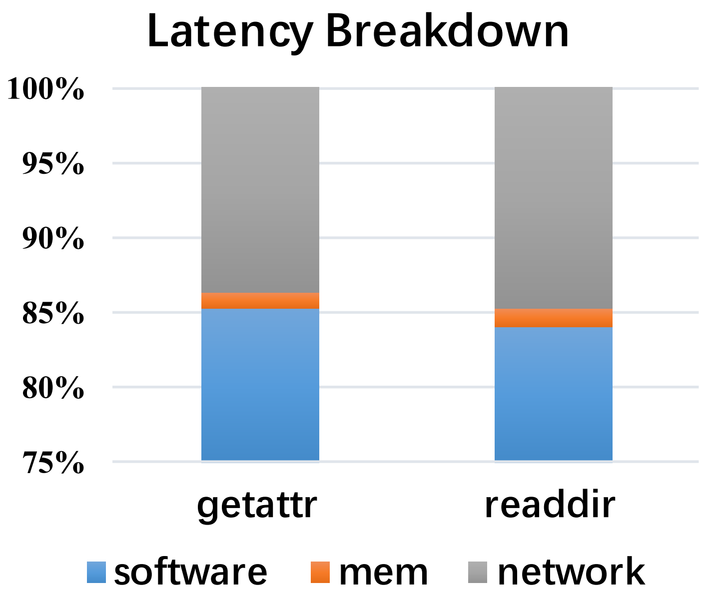
Evaluation Setup

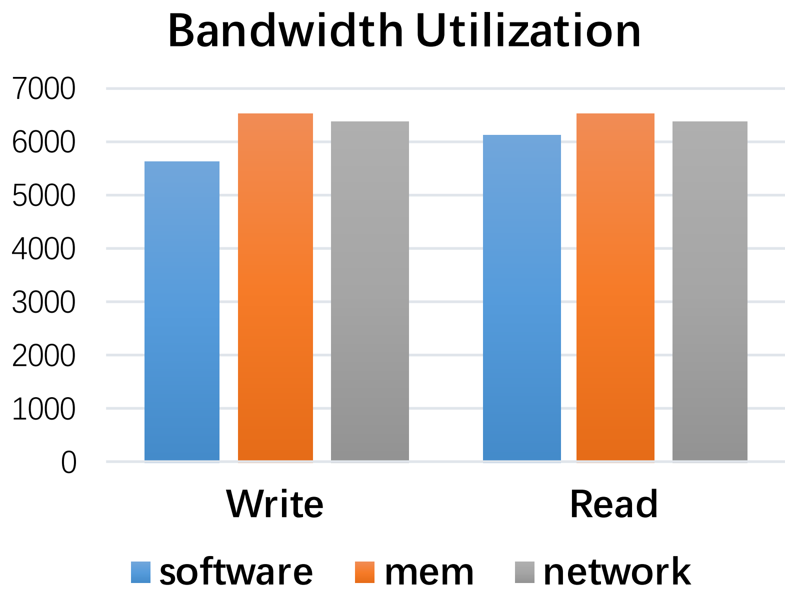
- Evaluation Platform

Cluster	CPU	Memory	ConnectX-3 FDR	Number
A	E5-2680 * 2	384 GB	Yes	* 5
B	E5-2620	16 GB	Yes	* 7

  - Connected with Mellanox SX1012 switch
- Evaluated Distributed File Systems
  - memGluster, runs on memory, with RDMA connection
  - NVFS[OSU], Crail[IBM], optimized to run on RDMA
  - memHDFS, Alluxio, for big data comparison

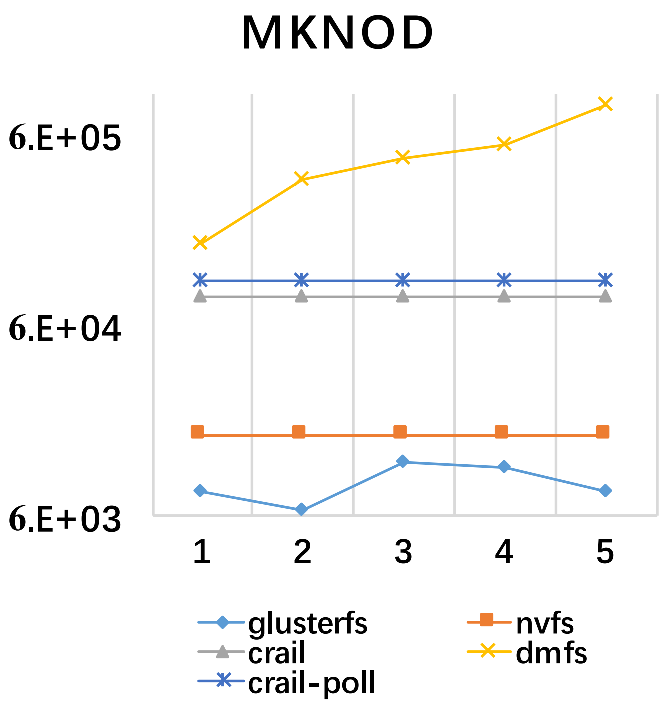
Overall Efficiency

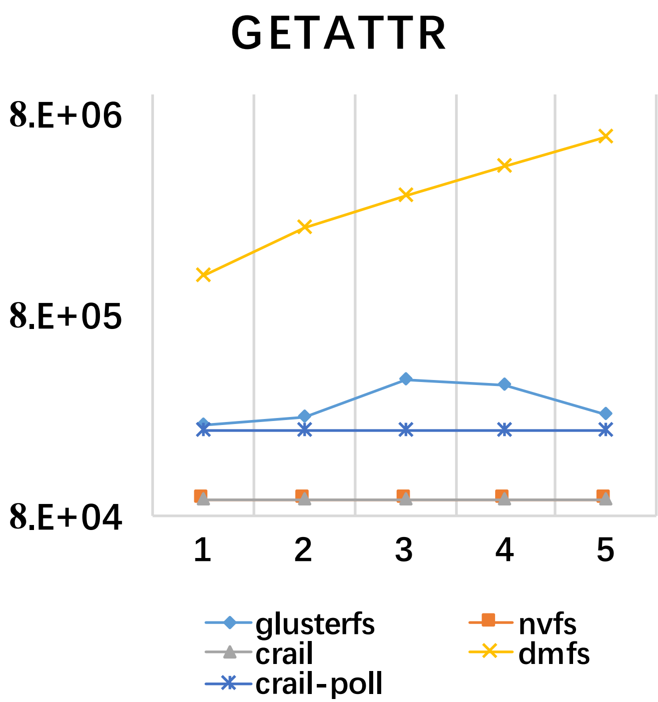


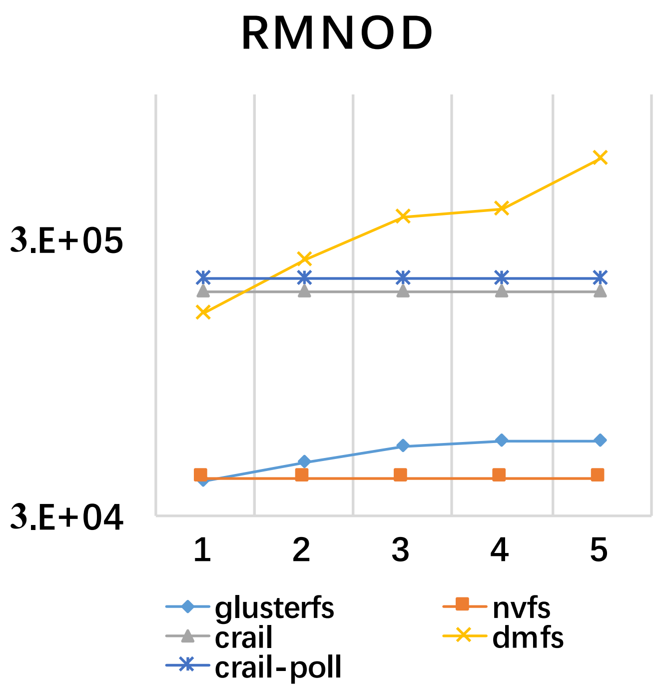


- Software latency is reduced to 6 us (85% of the total latency)
- Achieves read/write bandwidth that approaches the raw storage and network bandwidth

Metadata Operation Performance

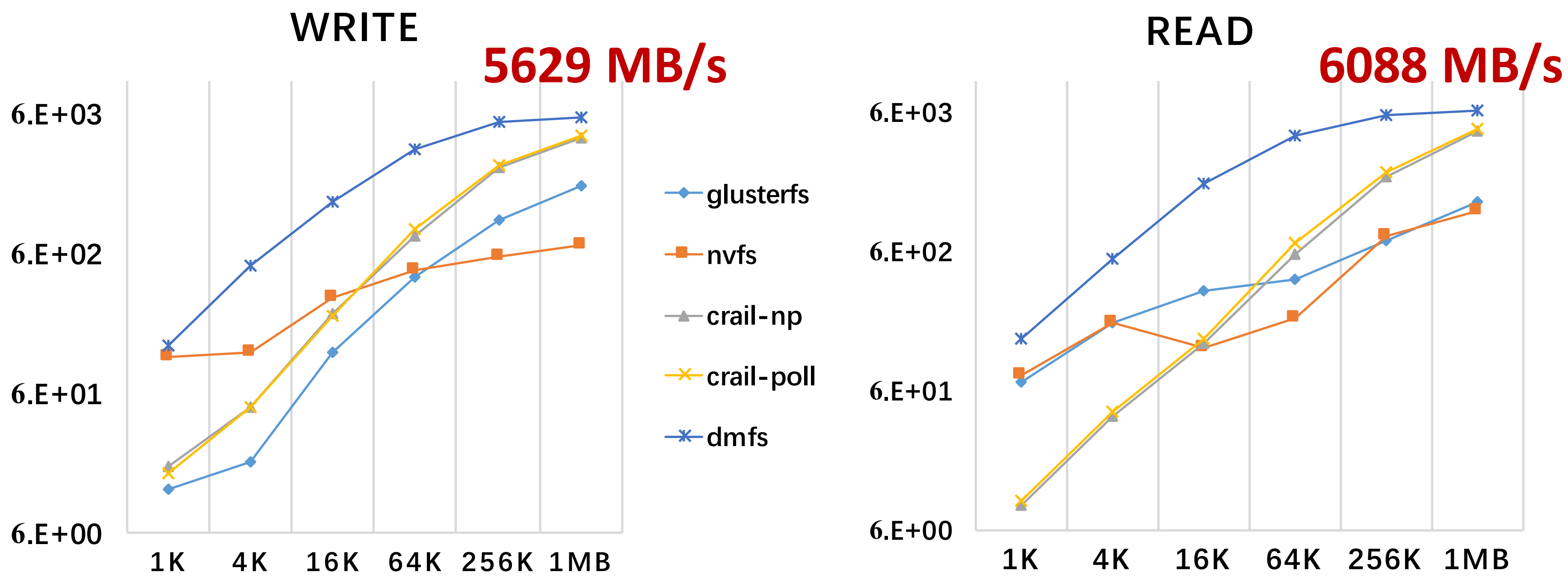






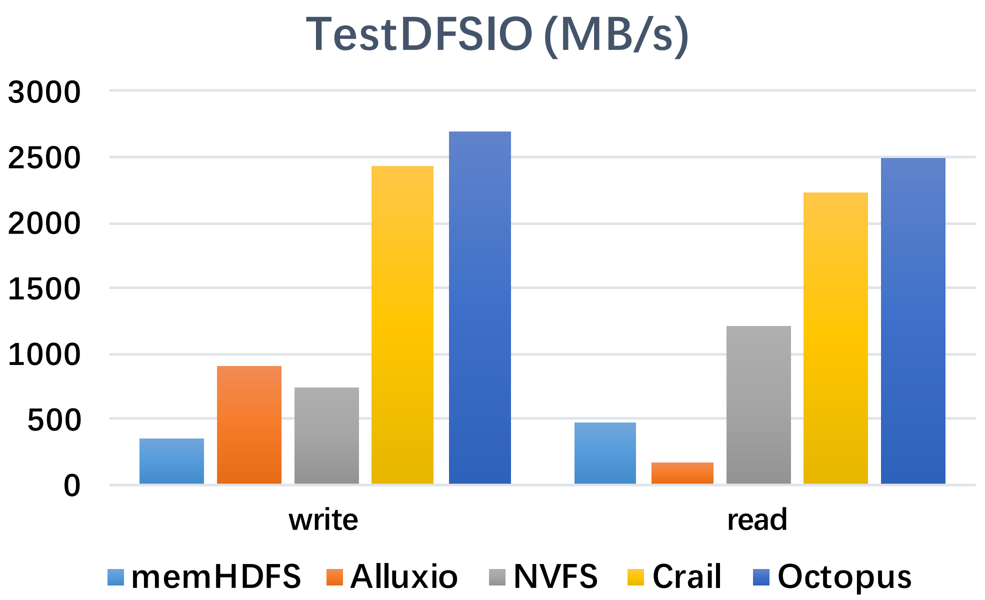
- Octopus provides metadata IOPS in the order of $10^5 \sim 10^6$
- Octopus can scales linearly

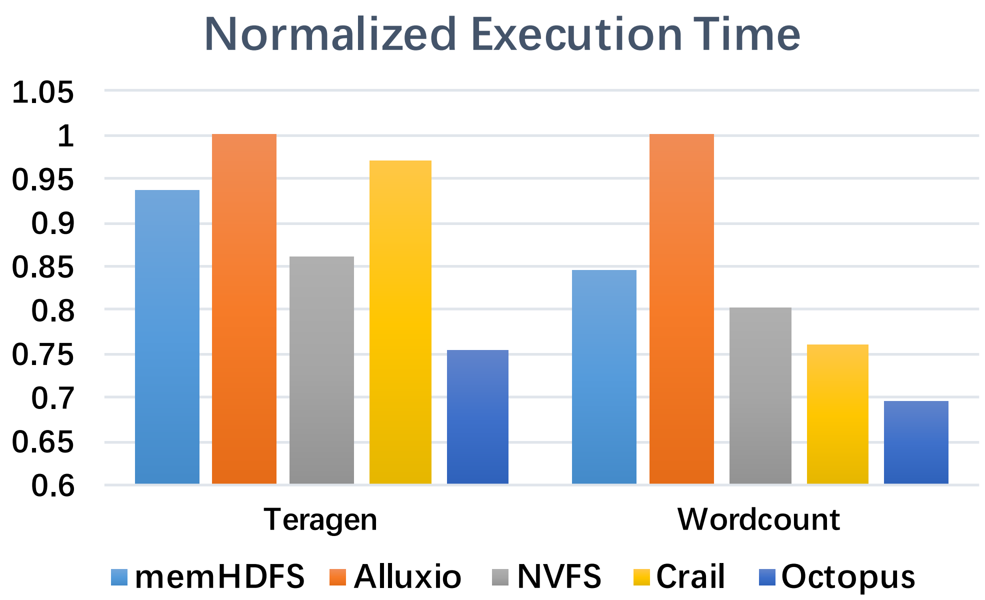
Read/Write Performance



- Octopus can easily reach the maximum bandwidth of hardware with a single client
- Octopus can achieve the same bandwidth as Crail even add an extra data copy [not shown]

Big Data Evaluation





- Octopus can also provide better performance for big data applications than existing file systems.

Conclusion

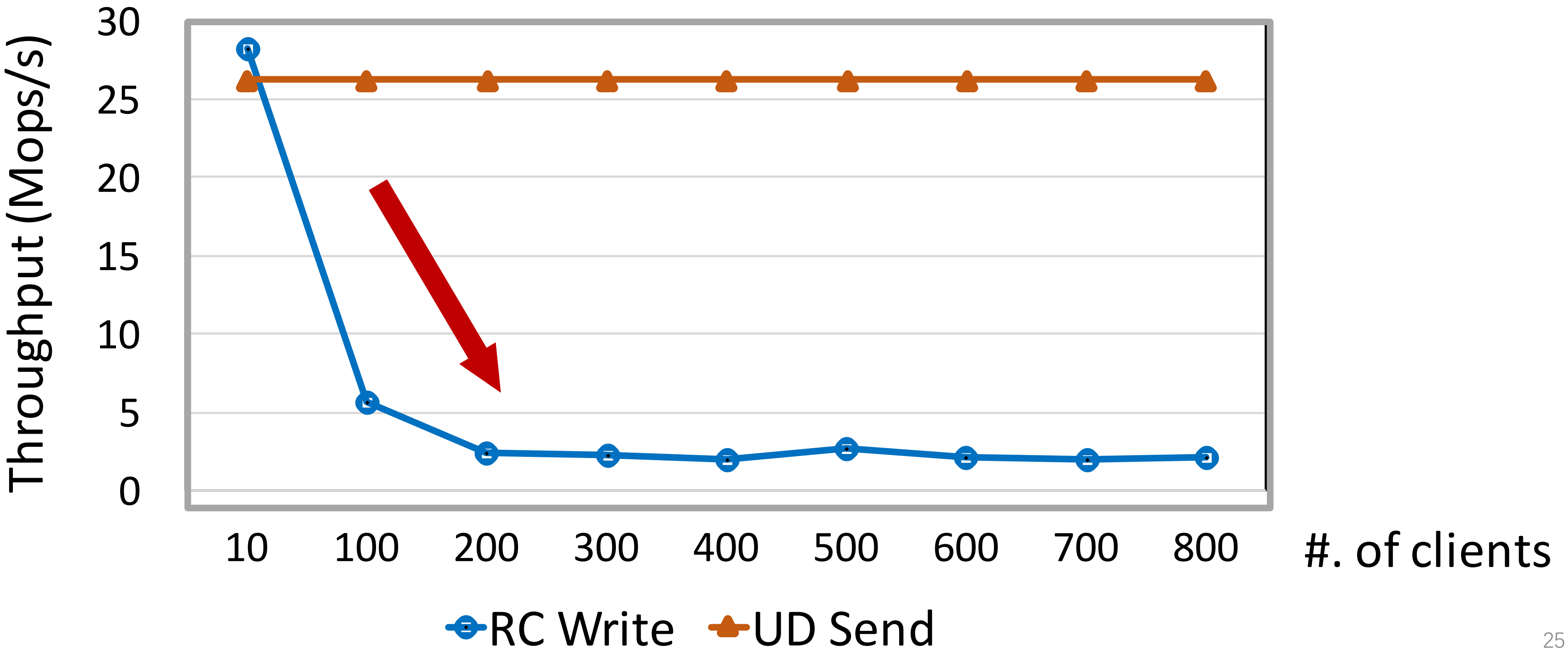
- Octopus provides high efficiency by redesigning the software
- Octopus's internal mechanisms
  - Simplifies data management layer by **reducing data copies**
  - **Rebalances network and server loads** with Client-Active I/O
  - Redesigns the **metadata RPC** and **distributed transaction** with RDMA primitives
- Evaluations show that Octopus significantly outperforms existing file systems

Outline

- Octopus: a RDMA-enabled Distributed Persistent Memory File System
  - Background and Motivation
  - Octopus Design
  - Evaluation
  - Conclusion
- Scalable and Reliable RDMA

RC is hard to scale

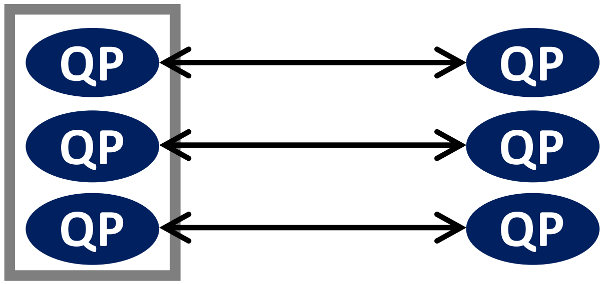
- MCX353A **ConnectX-3** FDR HCA (single port)
- **1** server node send verbs to **11** client nodes



RC vs UD

Reliable Connection (RC)

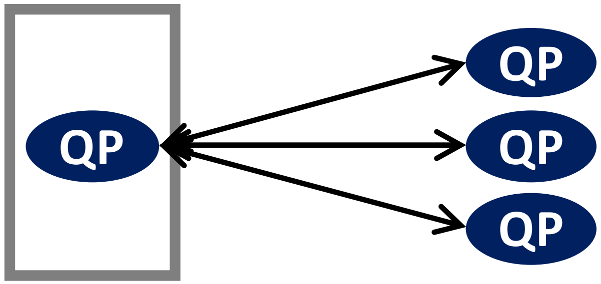
One-to-one paradigm



- Offloading with one-sided verbs
- Higher performance
- Reliable
- Flexible-sized transferring
- **Hard to scale**

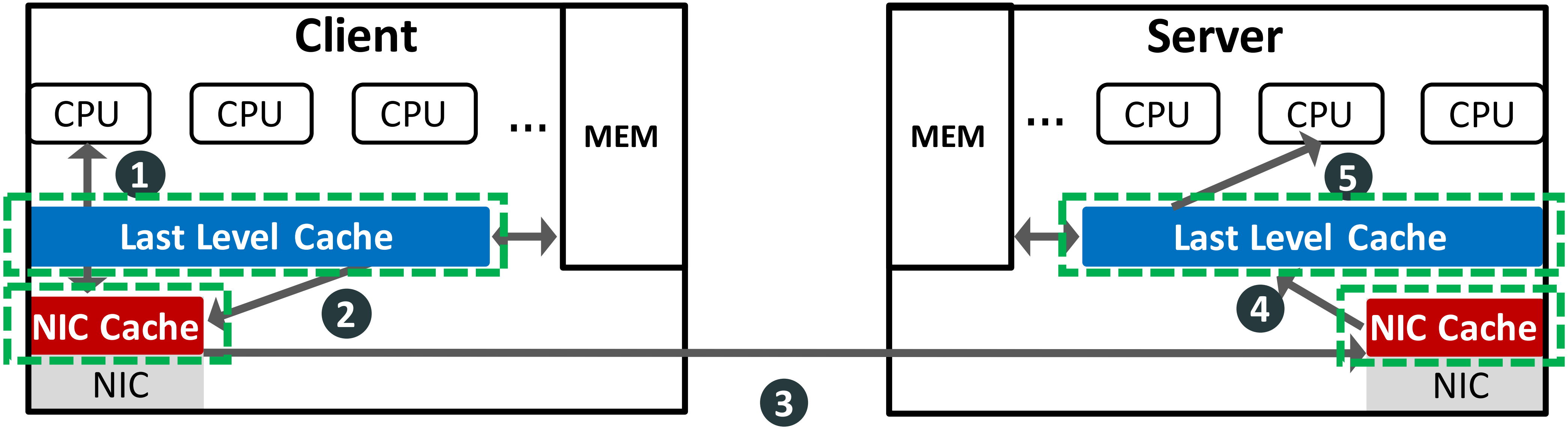
Unreliable Datagram (UD)

One-to-many paradigm



- Unreliable (risk of packet loss, out-of-order, etc.)
- Cannot support one-sided verbs
- MTU is only 4KB
- **Good scalability**

Why is RC hard to scale?

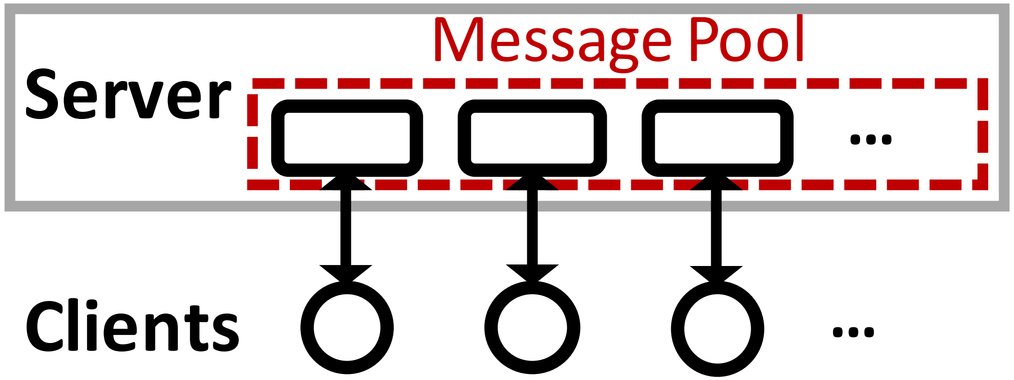


1 Memory-Mapped I/O
2 PCIe DMA Read
3 Packet Sending
4 PCIe DMA Write (DDIO enabled)
5 CPU Polls Message

Why is RC hard to scale?

Two types of Resource Contention:

- **NIC Cache**[1]
  - Mapping table
  - QP states
  - Work queue elements
- **CPU Cache**
  - DDIO writes data to LLC
  - Only **10%** reserved for DDIO

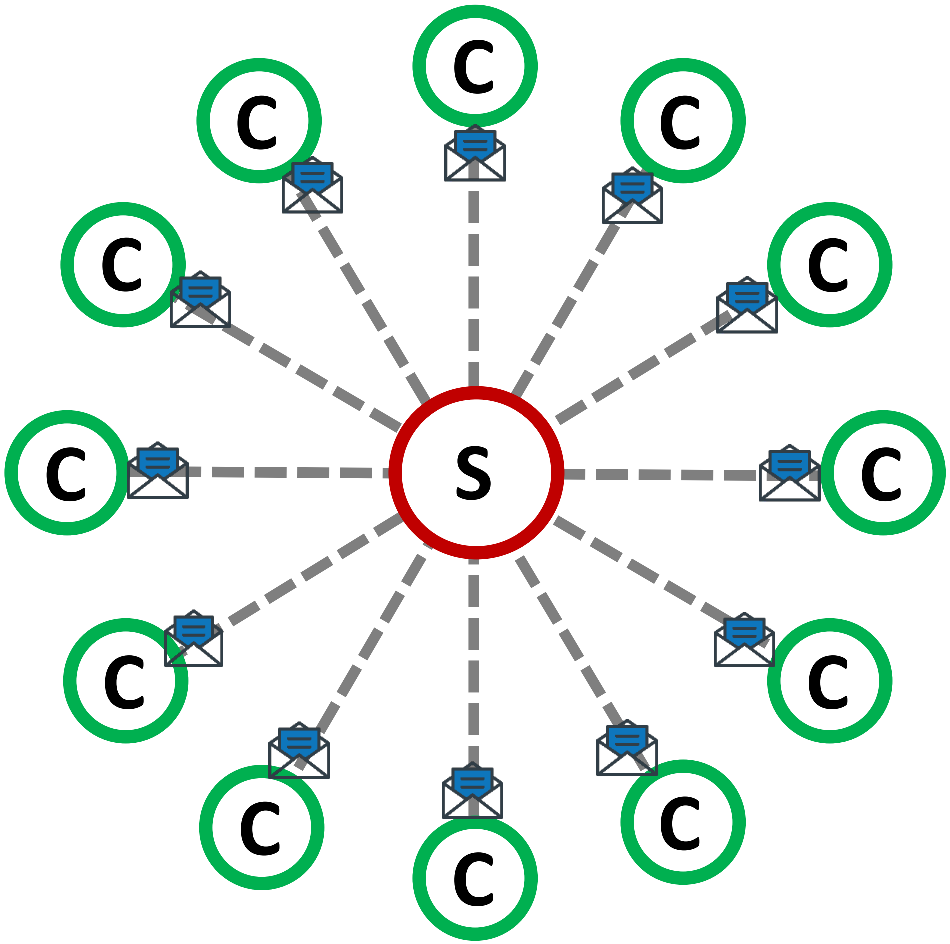


With **RC**, the size of cached data is **proportional** to the number of clients!

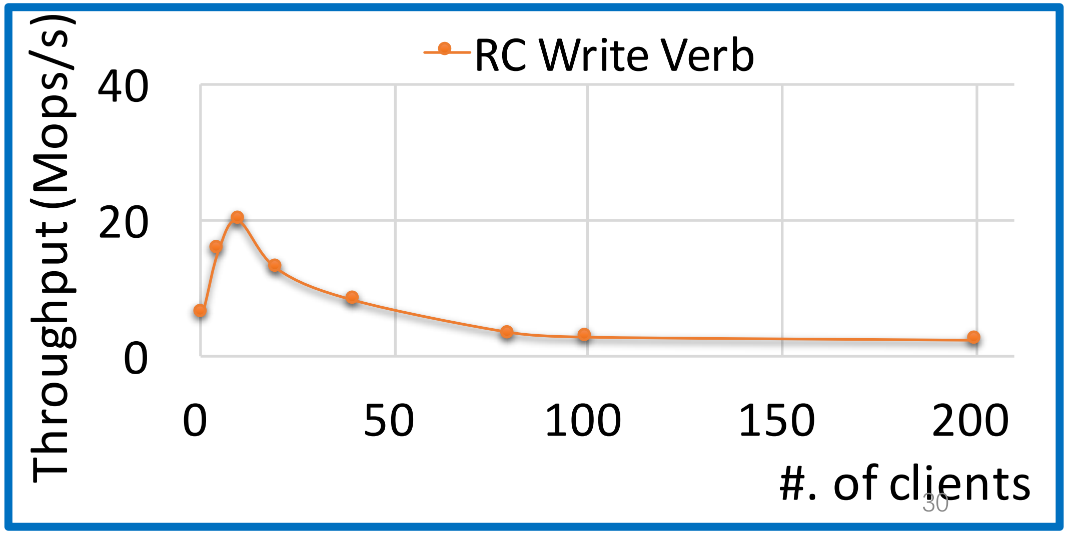
Our goal: how to make RC scalable

- **Focus on RPC primitive with RC write**
  - RPC is a good abstraction, widely used
  - RC write (one-sided) has higher throughput (FaRM)
- **Target at one-to-many data transferring paradigm**
  - e.g., MDS, KV store, parameter server, etc.
- **System-level solution**
  - Without any modifications to the hardware
- **Deployments**
  - Metadata server in Octopus
  - Distributed transactional system

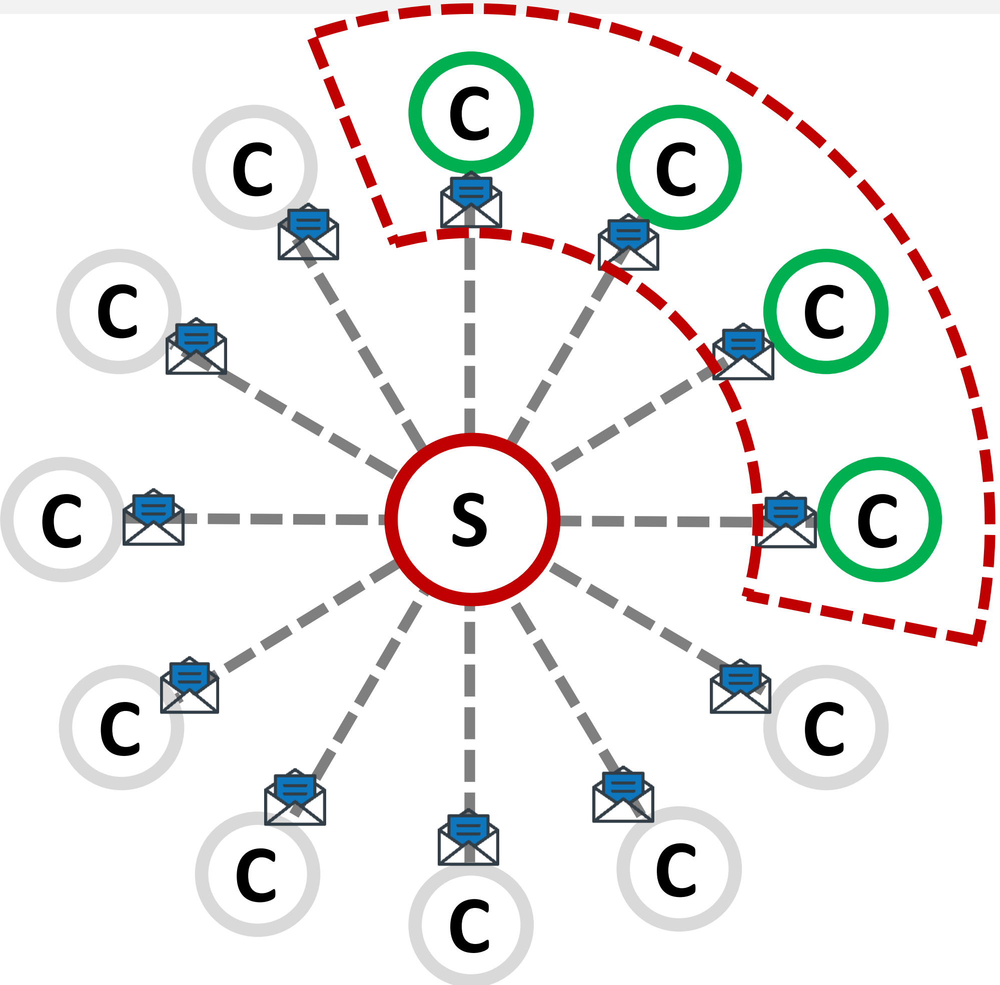
1. Grouping the connections



- **Naïve Approach**
  - NIC cache thrashing when the number of clients increases
  - Frequent swap in/out
  - Causing higher PCIe traffic

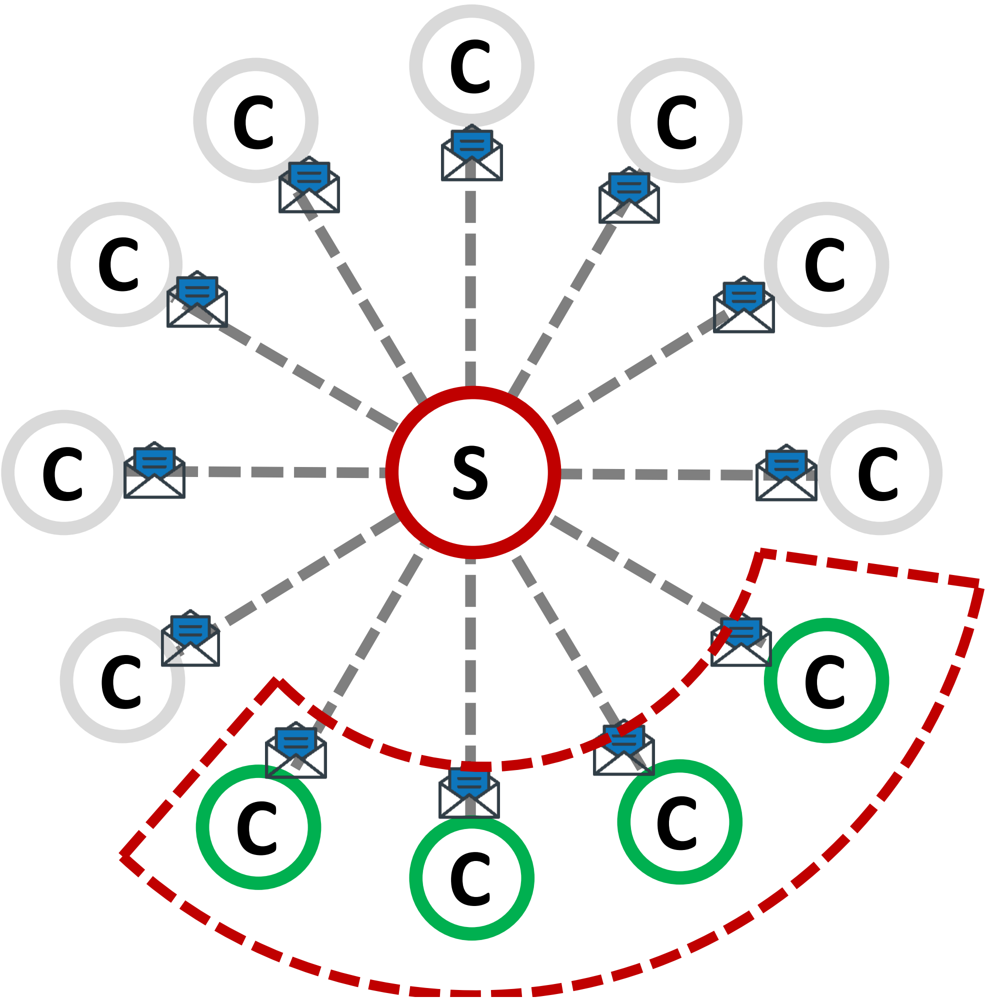


1. Grouping the connections



- **Connection Grouping**
  - Serve **one group at a time slice**

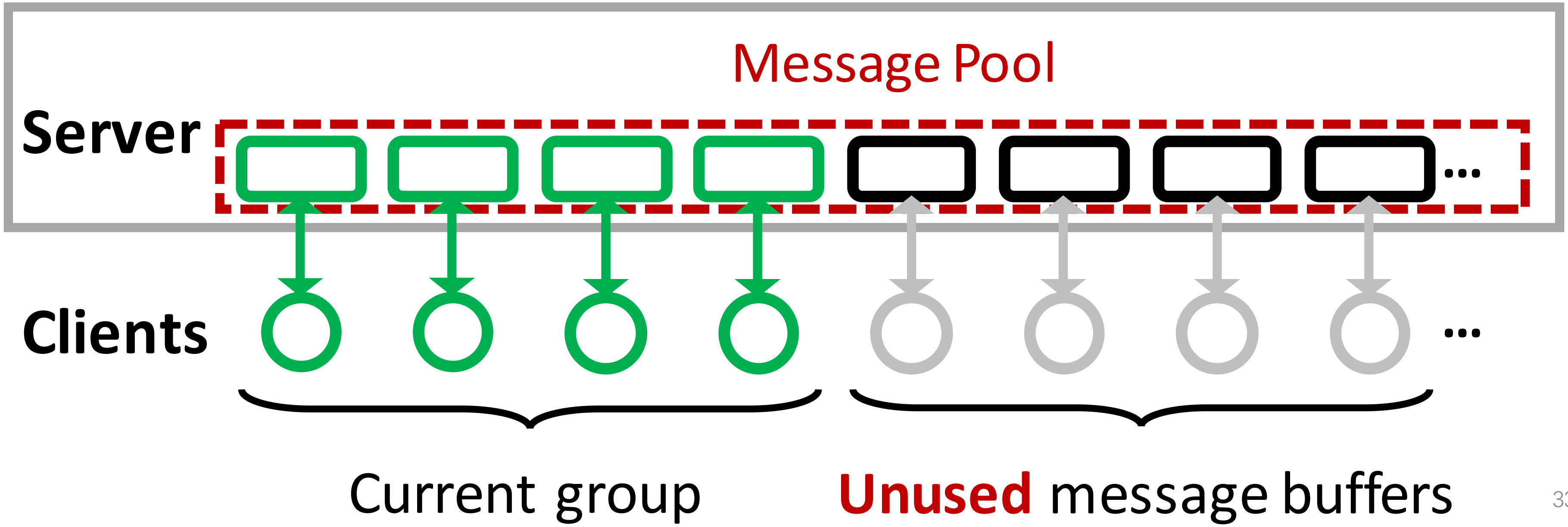
1. Grouping the connections



- **Connection Grouping**
  - Serve **one group at a time slice**
  - **Better cache locality:** recently accessed metadata is more likely be used again

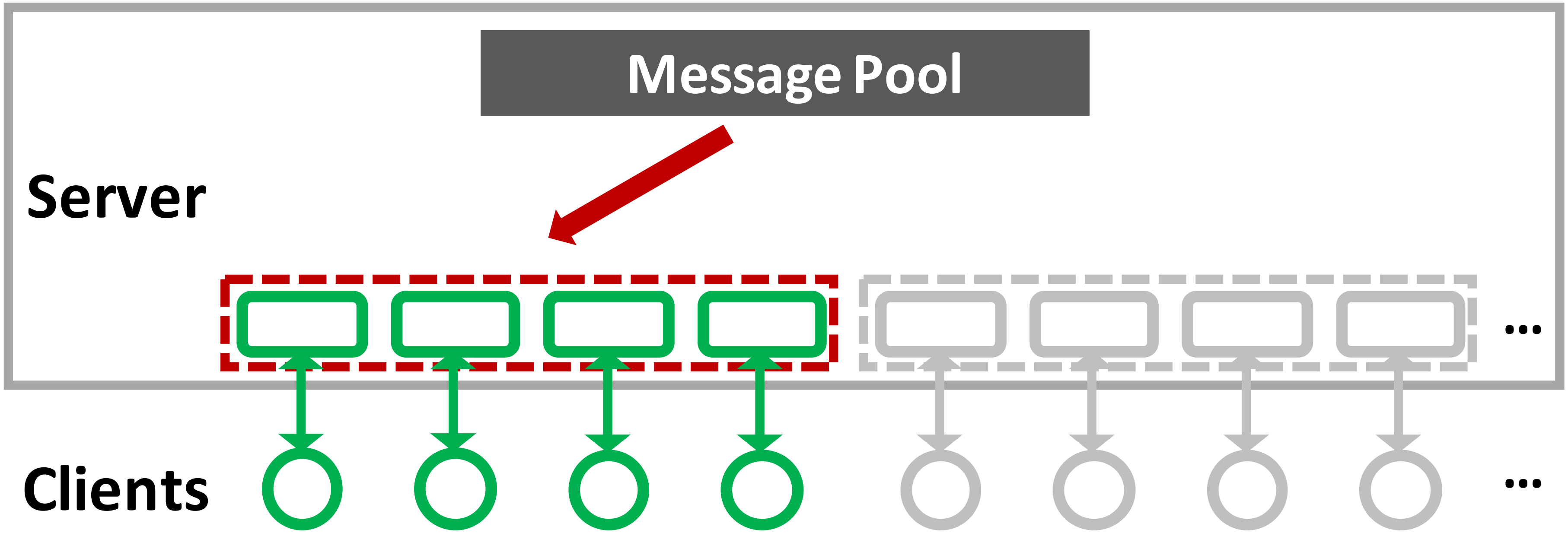
2. Virtualized Mapping

- **Alleviate the contention in the CPU cache**
  - Reduce memory footprint in the message pool
- **Observations:**
  - When grouping the clients, only **part** of the message pool is used



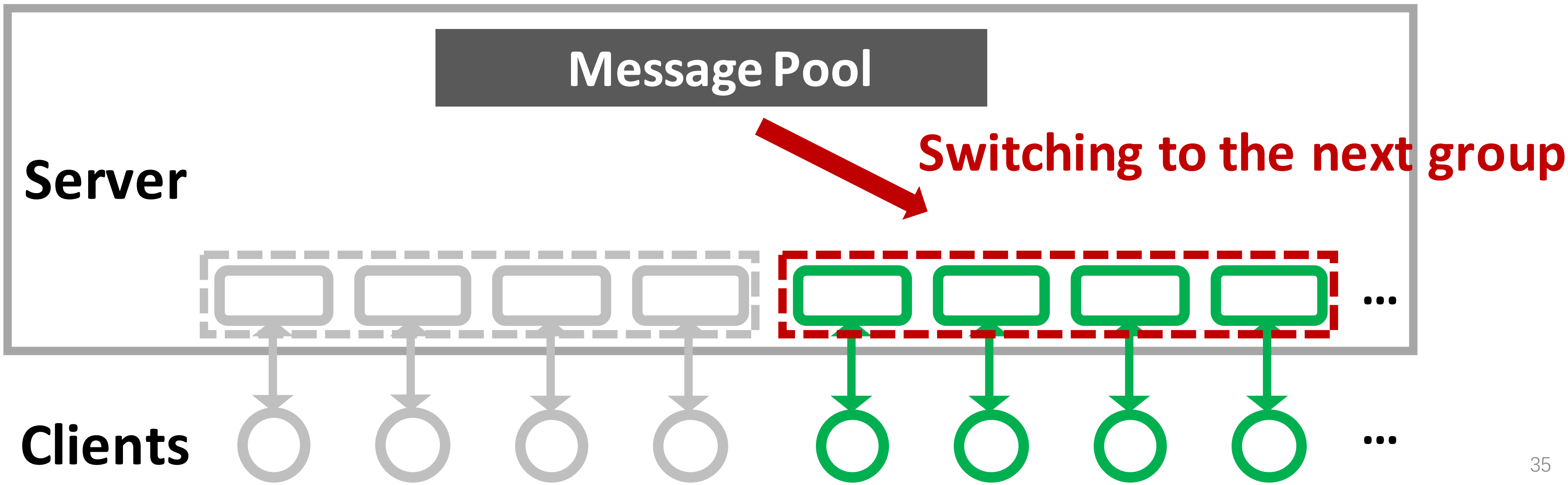
2. Virtualized Mapping

- We don’t need to assign a message buffer for each client
  - **Virtualize** a single physical message pool to be **shared** among multiple groups
  - Without extra overhead for loading/saving the context



2. Virtualized Mapping

- We don't need to assign a message buffer for each client
  - **Virtualize** a single physical message pool to be **shared** among multiple groups
  - Without extra overhead for loading/saving the context



Other Challenges & solutions

- **Static grouping** is **suboptimal** when clients have
  - Varying requirements for the tail latency
  - Varying frequencies of the posted RPCs
  - Varying payload sizes
  - Varying execution times for different handlers

➡ **Priority-based scheduler:** monitors the performance of each clients and dynamically adjust the group size and time slice length.

- **Switching** between the groups should be **efficient**

➡ **Warmup pool:** before being served, clients from the next group put their new requests in the warmup pool first

Details in [2] *Scalable RDMA RPC on Reliable Connection with Efficient Resource Sharing*. Youmin Chen, Youyou Lu, Jiwu Shu, in **Eurosys'19**

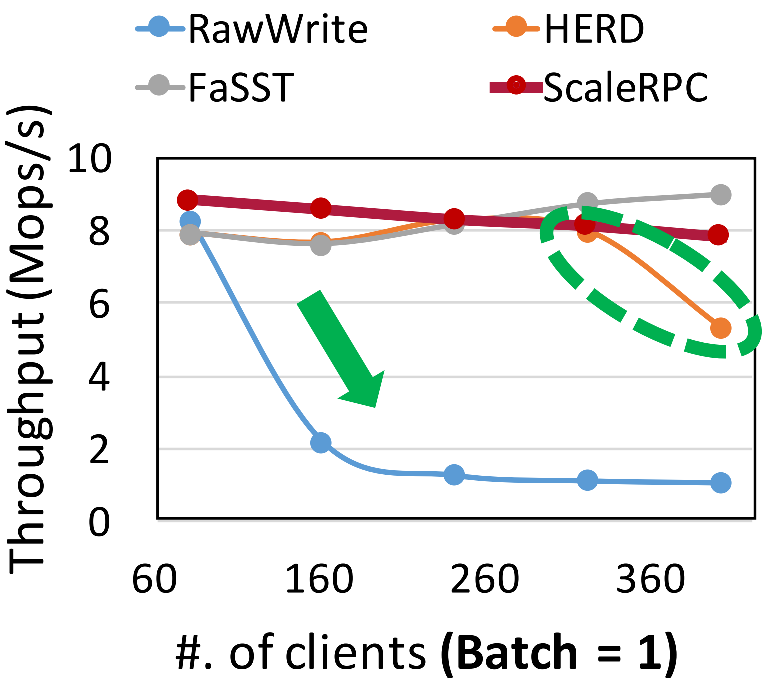
Evaluation

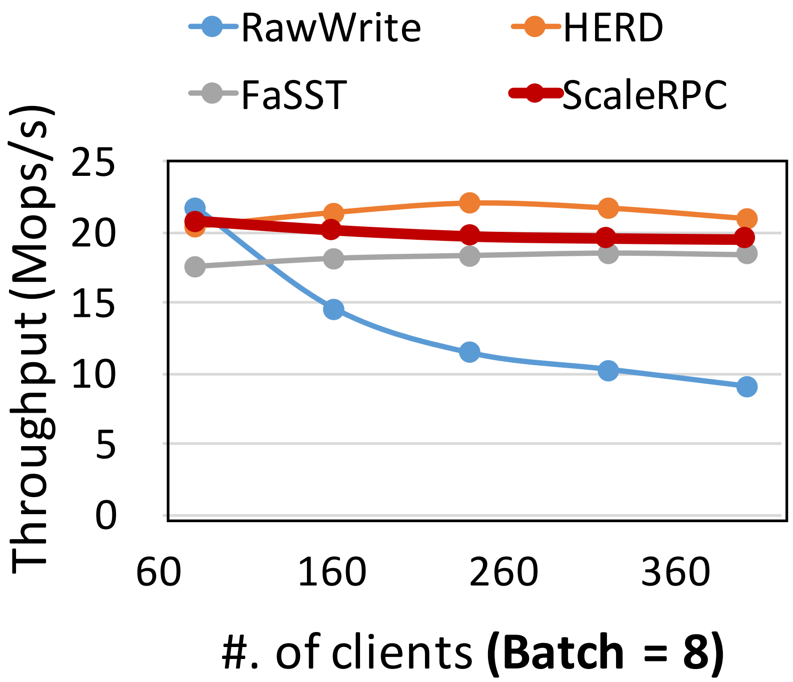
- **Platform**
  - 2× 2.2GHz Intel Xeon E5-2650 v4 CPUs (24 cores in total)
  - 128 GB DRAM
  - MCX353A CX-3 FDR HCAs (56 Gbps IB and 40 GbE)
  - 12-node cluster connected with Mellanox SX-1012 switch
- **Compared Systems**

RPC	Description
RawWrite RPC	A baseline RPC with all the optimizations in ScaleRPC disabled
HERD RPC	A scalable RPC with a hybrid of UC write and UD send verbs
FaSST RPC	A scalable RPC based on UD send verbs

Evaluation

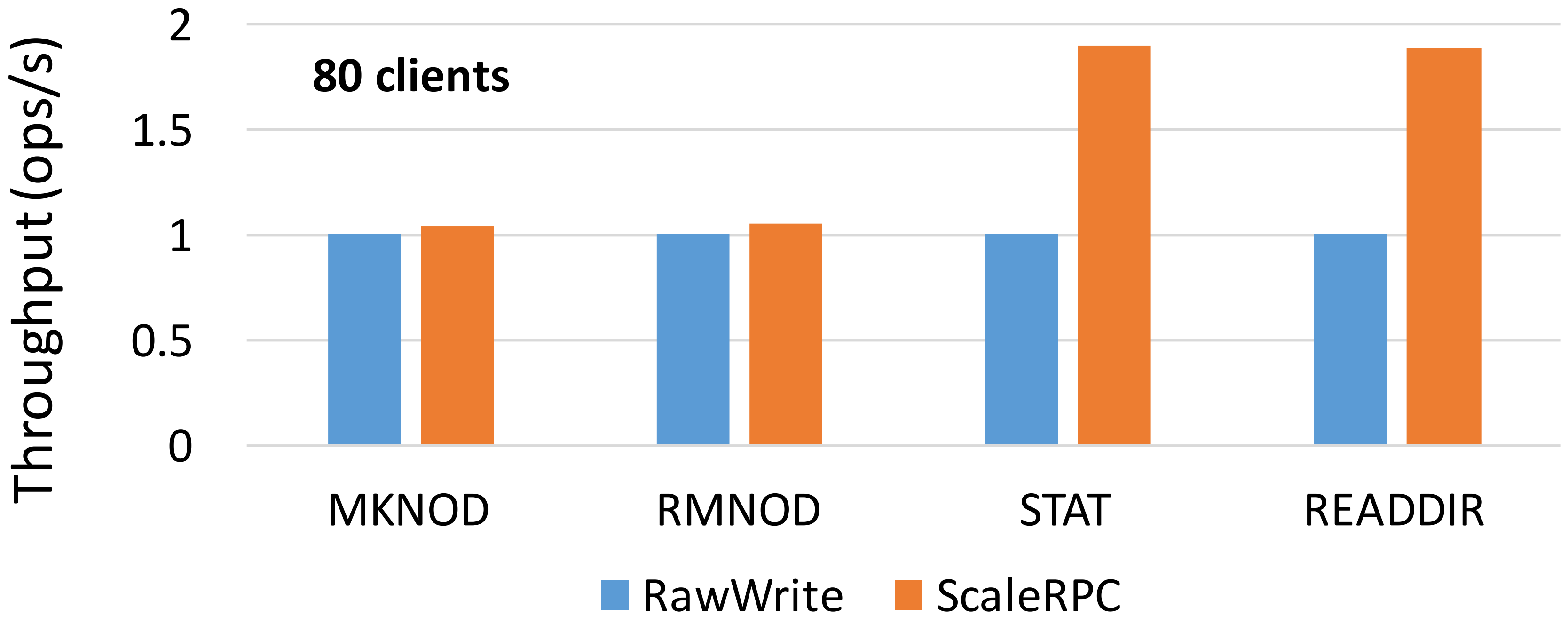
■ Throughput





Evaluation

- **Metadata Server in Octopus (Distributed File System)**



Thanks

[1] *Octopus: an RDMA-enabled Distributed Persistent Memory File System*. Youyou Lu, Jiwu Shu, Youmin Chen, Tao Li, in **USENIX ATC'17**

[2] *Scalable RDMA RPC on Reliable Connection with Efficient Resource Sharing*. Youmin Chen, Youyou Lu, Jiwu Shu, in **Eurosys'19**