


FaSST: Fast, Scalable, and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs

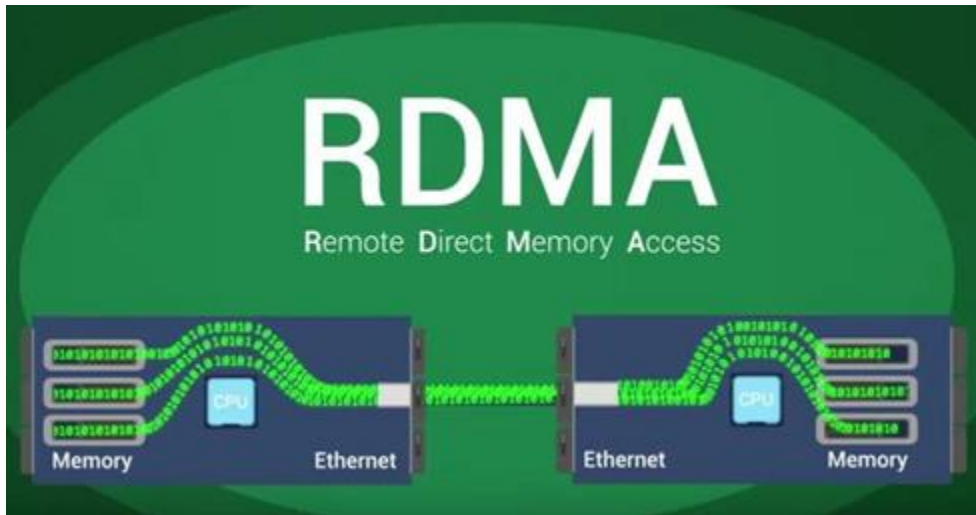


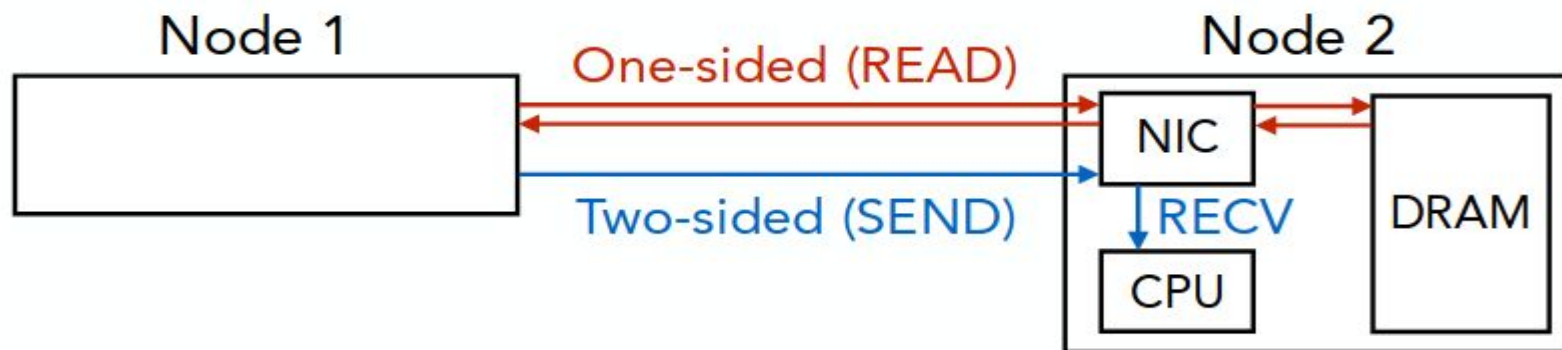
Anuj Kalia (CMU), Michael Kaminsky (Intel Labs), David Andersen (CMU)

RDMA

RDMA is a network feature that allows direct access to the memory of a remote computer

- Modes of communication
 - One-sided RDMA (CPU bypass)
 - Read
 - Write
 - Fetch_and_add
 - Compare_and_swap.
 - An MPI with SEND/RECV verbs
 - Remote CPU is used





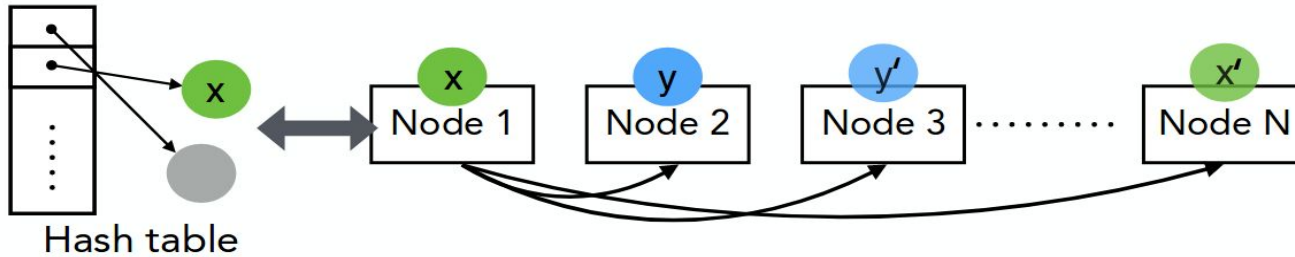
Existing systems

Use one-sided RDMA (READs and WRITEs) for transactions

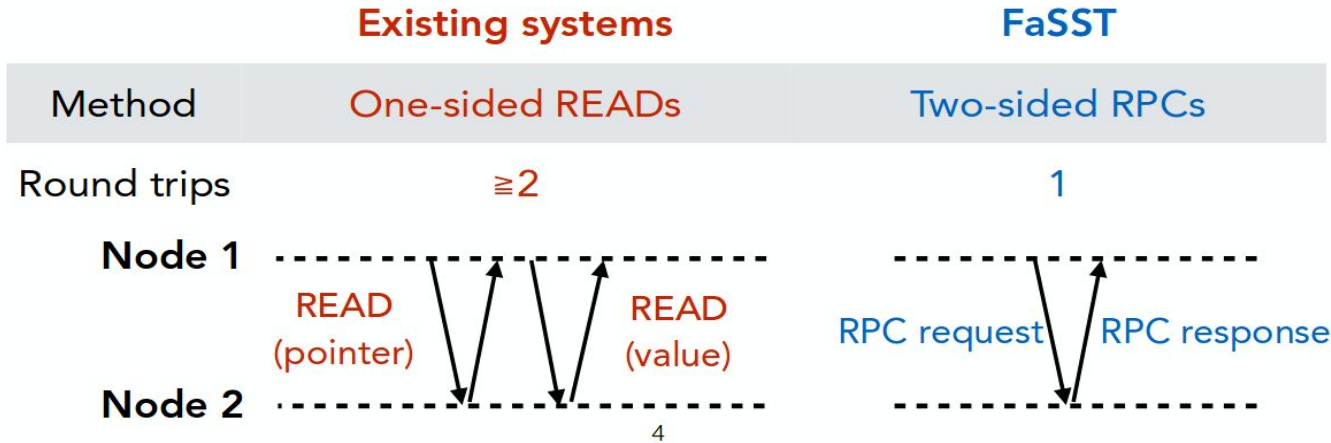
FaSST

- Uses RPCs over two-sided ops
- ~2x faster than existing systems
- Fast, scalable, simple

Transaction environment



How to access remote data structures?



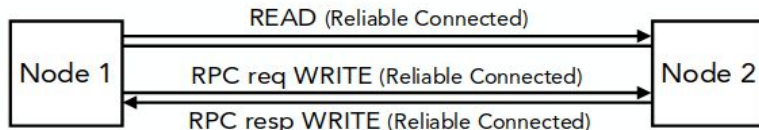
*slide taken from author's presentation at OSDI'16

Problem with one-sided RDMA

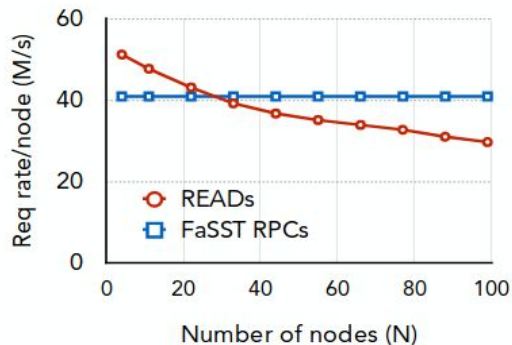
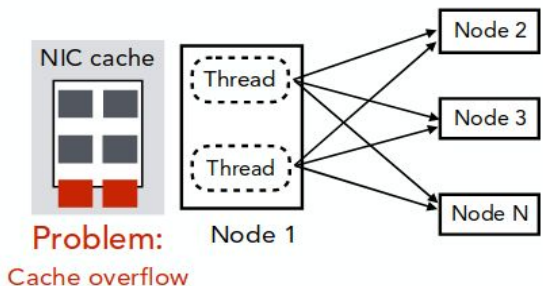
One-sided RDMA does not scale

READs & WRITEs must use a connected transport layer

One-sided systems

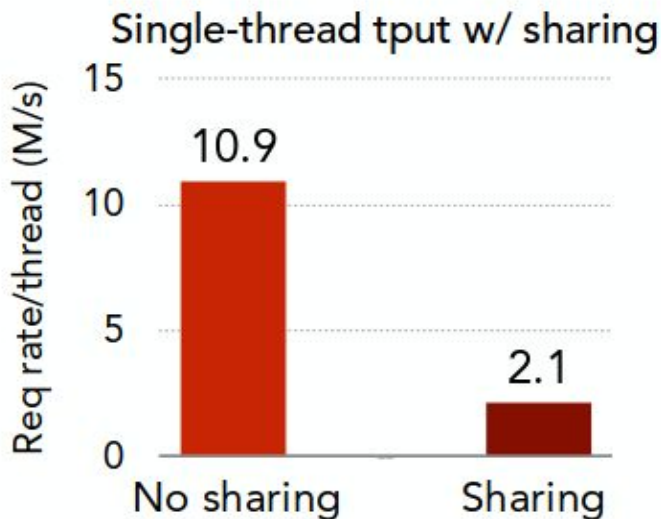
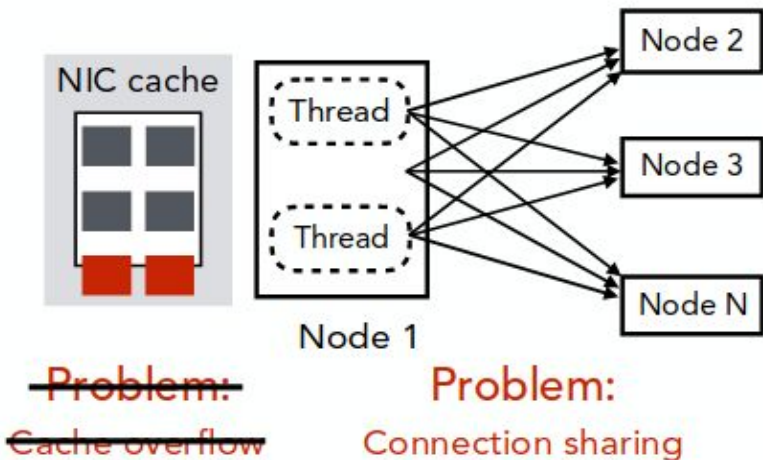


**Solution-
Connection sharing**



Problem with one-sided Reads

CPU overhead of connection sharing



Locking overheads

Existing systems

FaSST

Method

One-sided READs

Two-sided RPCs

Round trips

≥ 2

1

Scalable transport



Effect: NIC cache misses

Lock-free I/O

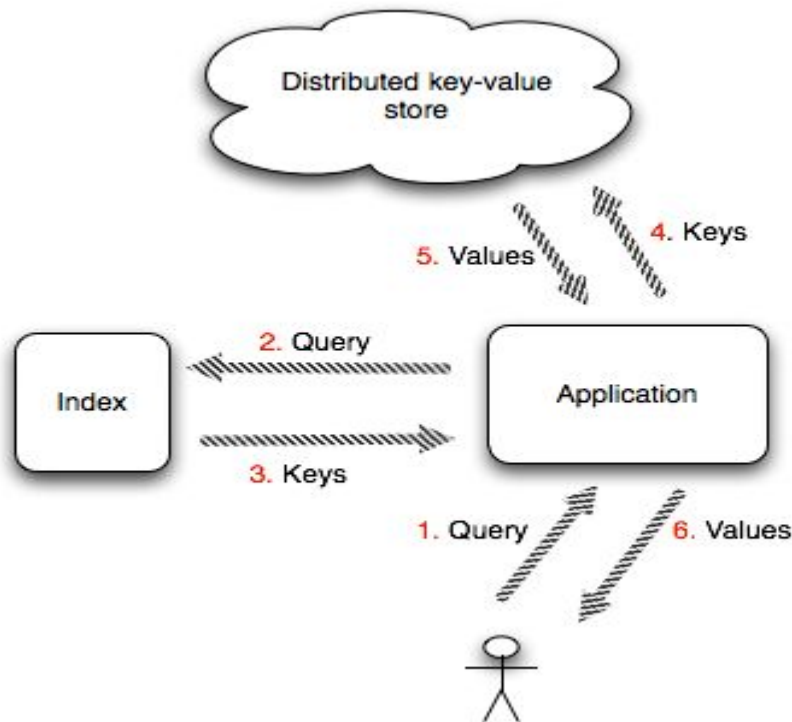


Effect: Low per-thread tput

Contribution

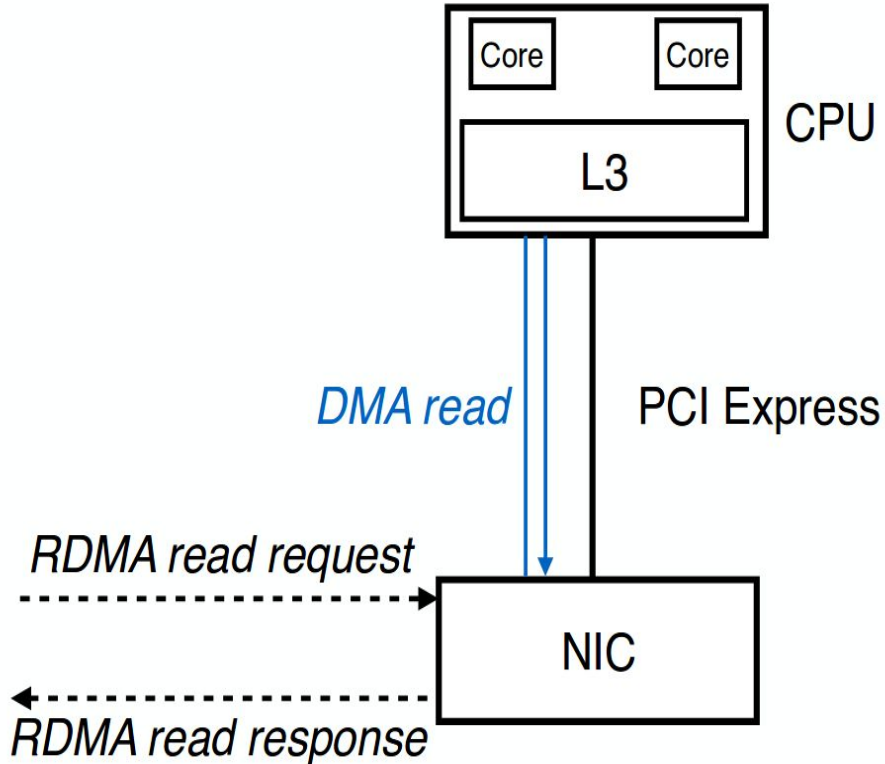
- FaSST : In-memory distributed transaction processing system based on RDMA
 - RDMA-based system for key-value store
 - RPC style mechanism implemented over unreliable datagrams
 - In-memory transactions
 - Serializability
 - Durability
 - Better scalability
- Existing RDMA-based transaction processing
 - One-sided RDMA primitives
 - Flexibility and scalability issues
 - Bypassing the remote CPU

Distributed key-value store



- **Multiple RDMA Reads** to fetch the value
 - One read to get the pointer from the index
 - One read to get the actual data
 - **Solutions**
 - Merge the data with index [FaRM]
 - Caching the index at all servers

RDMA

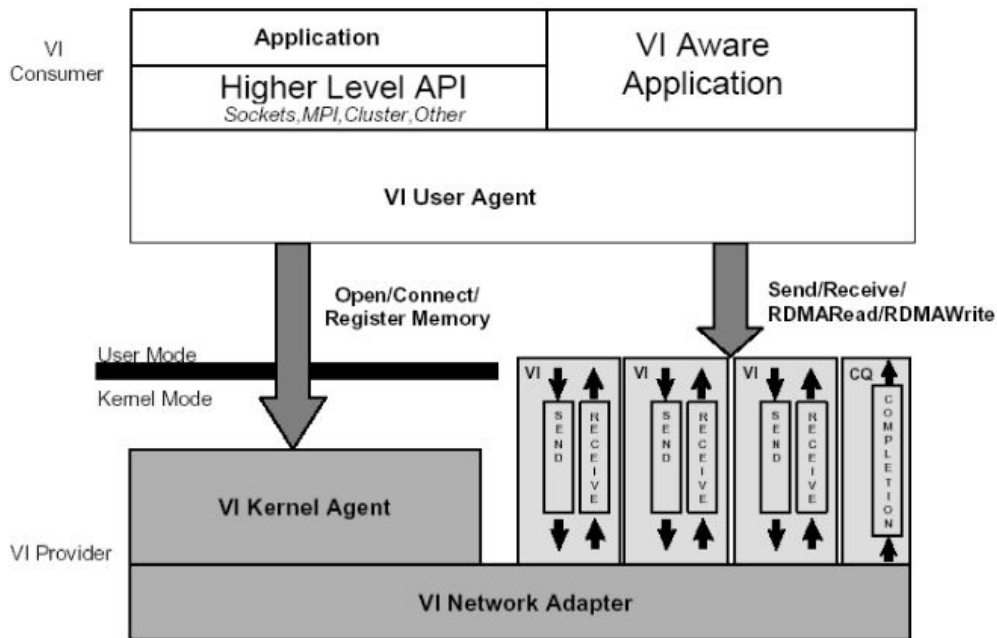


RDMA operations

- Remote CPU bypass (one-sided)
 - Read
 - Write
 - Fetch-and-add
 - Compare-and-swap
- Remote CPU involved (messaging, two-sided)
 - Send
 - Recv

VIA-based RDMA

VI Architectural Model



- User level, zero-copy networking
- Commodity RDMA implementations
 - InfiniBand
 - RoCE
- Connection oriented or connection less

VIA-based RDMA

- Facilitates fast and efficient data exchange between applications running on different machines
- Allows applications(VI consumers) to communicate directly with the network card(VI provider) via common memory areas bypassing the OS
- Virtual interfaces are called queue pairs
 - Send queue
 - Receive queue
- Applications access QPs by posting verbs
 - Two-sided verbs, send and receive involve CPU
 - One-sided verbs, read, write and atomic bypass the CPU

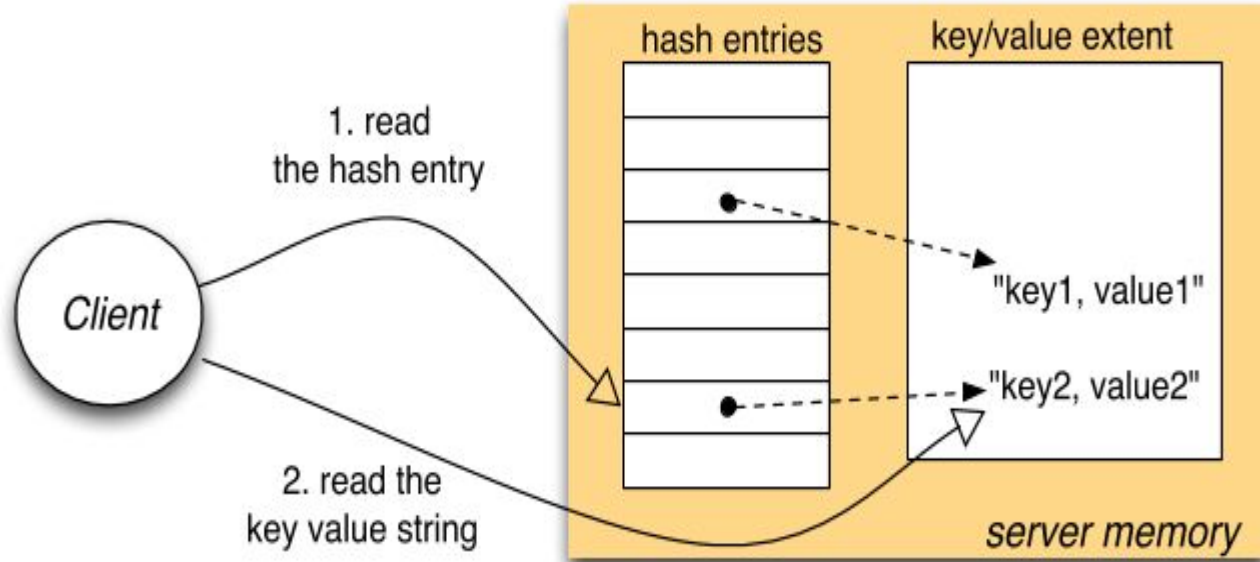
RDMA transports

- Connection oriented
 - One-to-one communication between two QPs
 - Thread creates N QPs to communicate with N remote machines
 - One-sided RDMA
 - End-to-end reliability
 - Poor scalability due to limited NIC memory
- Connectionless
 - One QP communicates with multiple QPs
 - Better scalability
 - One QP needed per thread

RDMA transports

- Reliable
 - In-order delivery of messages
 - Error in case of failure
- Unreliable
 - Higher performance
 - Avoids ACK packets
 - No reliability guarantees
- Modern high speed networks
 - Link layer provides reliability
 - Flow control for congestion-based losses
 - Retransmission for error-based losses

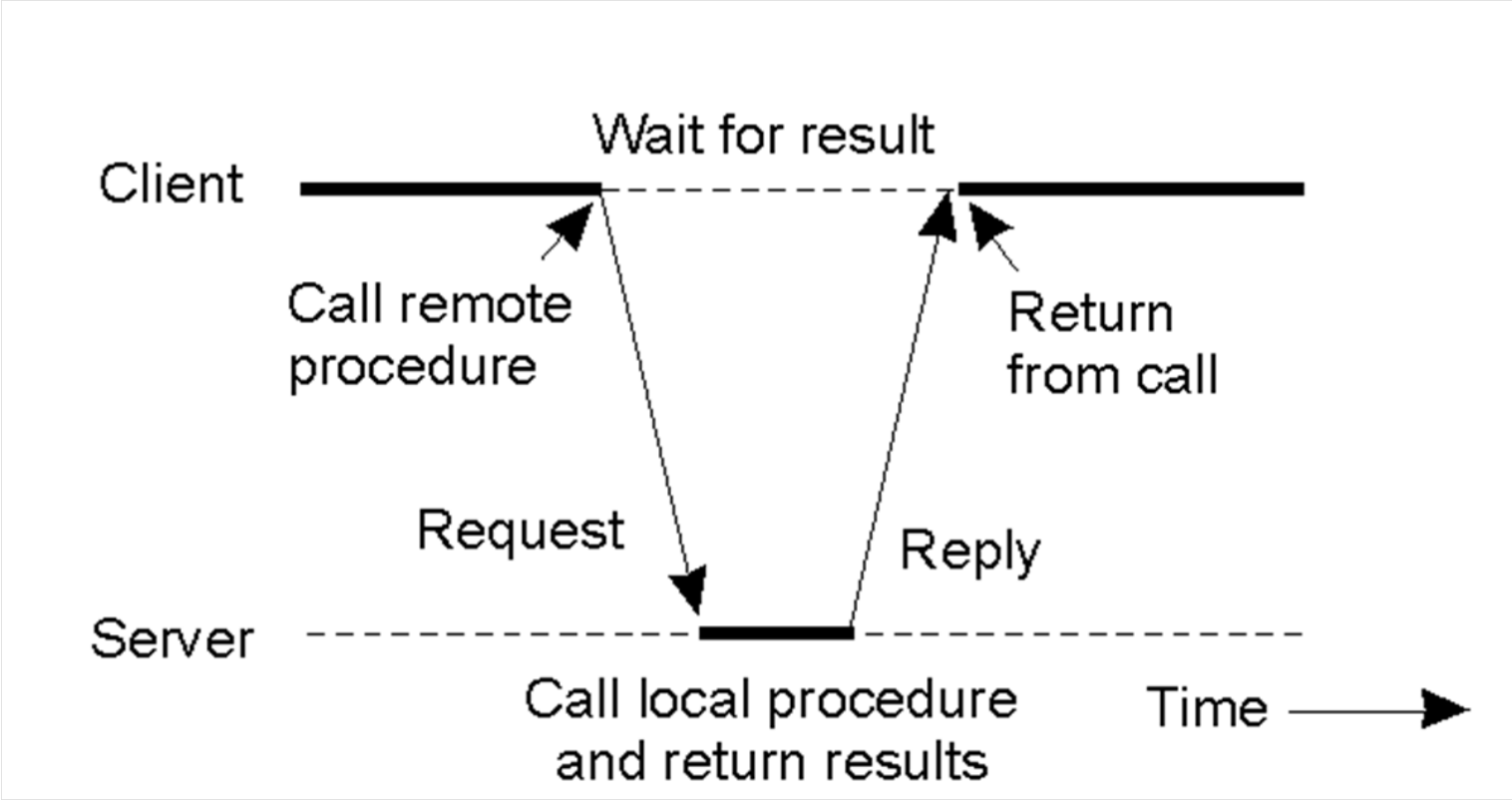
One-sided RDMA



One-sided RDMA for transaction processing system

- Saves remote CPU cycles
- Remote reads, writes, atomic operations
- Connection-oriented nature
- Drawbacks
 - Two or more RDMA reads to access data
 - Lower throughput & higher latency
 - Sharing local NIC queue pairs

RPC



RPC over two-sided datagrams verbs

- Remote CPU is involved
- Data is accessed in a single round trip
- FaSST is an all-to-all RPC system
 - Fast
 - 1 round trip
 - Scalable
 - One QP per core
 - Simple
 - Remote bypassing designs are complex, redesign and rewrite data structures
 - RPC based designs are simple, reuse the existing data structures
 - CPU-efficient



FaSST

Uses RPC as opposed to READs in
one-sided RDMA

Uses datagram as opposed to
connection oriented transport

Advantages of RPCs over one-sided RDMA

- Recent work focused on using one-sided RDMA primitives
 - Clients access remote data structures in server's memory
 - One or more reads
 - Optimizations help reducing the number of reads
- Value-in-index
 - Used in FaRM
 - Hash table access in 1 READ on avg
 - Specialized index to store data adjacent to its index entry
 - Data read along with the index
 - Limitation
 - Read amplification by a factor of 6-8x
 - Reduced throughput

Advantages of RPCs over one-sided RDMA

- Caching the index
 - Used in DrTM
 - Index of hash table cached at all servers in the cluster
 - Allows single READ GETs
 - Works well for high locality workloads
 - But indexes can be large e.g. OLTP benchmarks
- RPCs allows access to partitioned data stores with two messages-request and reply
 - No message amplification
 - No multiple round trips
 - No caching required
 - Only short RPC handlers

Advantages of datagram transport over connection-oriented transport

- Connection oriented transport
 - A cluster with N machines and T threads per machine
 - $N \cdot T$ QPs per machine
 - May not fit in NIC's QP cache
 - Share QPs to reduce QP memory footprint
 - Contention for locks
 - Reduced CPU efficiency
 - Not scalable
- QP sharing reduces per-core throughput of one-sided READs by up to 5.4x

Advantages of datagram transport over connection-oriented transport

- Datagram transport
 - One QP per CPU core to communicate with all remote cores
 - Exclusive access to QP by each core
 - No overflowing of NIC's cache
 - Connection less
 - Scalability due to exclusive access
 - Doorbell Batching reduces CPU use
- RPCs achieve up to 40.9 Mrps/machine

Doorbell Batching

- per-Qp doorbell register on the NIC
- Post operations(send/recv) by user processes to NIC
 - Write to doorbell register
 - PCIe involved hence expensive
 - Flushing the write buffers
 - Memory barriers for ordering
- PCIe messages are expensive
 - Reduce CPU-to-NIC messages (MMIOs)
 - Reduce NIC-to-CPU messages (DMAs)
- Doorbell batching reduces MMIOs

Doorbell Batching

- With one-sided RDMA reads
 - Multiple doorbell ringing required for a batch of packets
 - Connected QPs
 - Number of doorbells equal to number of message destinations appearing in the batch

- For RPCs over datagram transport
 - One doorbell ringing per batch
 - Regardless of individual message destinations
 - Lesser PCIe overheads

FaSST distributed transactions

- Distributed transactions in a single data centre
- A single instance scales to few hundred nodes
- Symmetric model
- Data partitioned based on a primary key
- In-memory transaction processing
- Fast userspace network I/O with polling
- Concurrency control, two phase commit, primary backup replication
- Doorbell batching

Setup

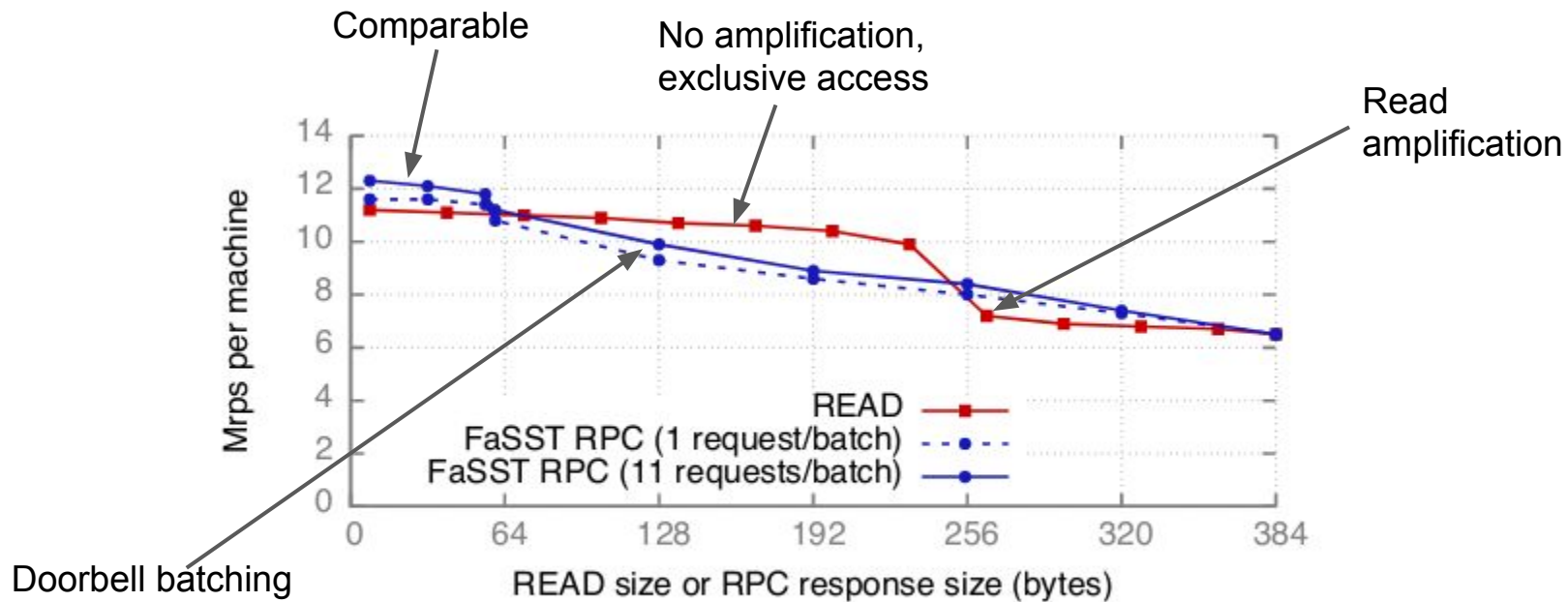
Cluster used	# nodes	# cores	NIC	
CX3	192	8	ConnectX-3	
CIB	11	14	Connect-IB	2x higher BW

Comparison of RPC and one-sided READ performance

Comparison on small cluster

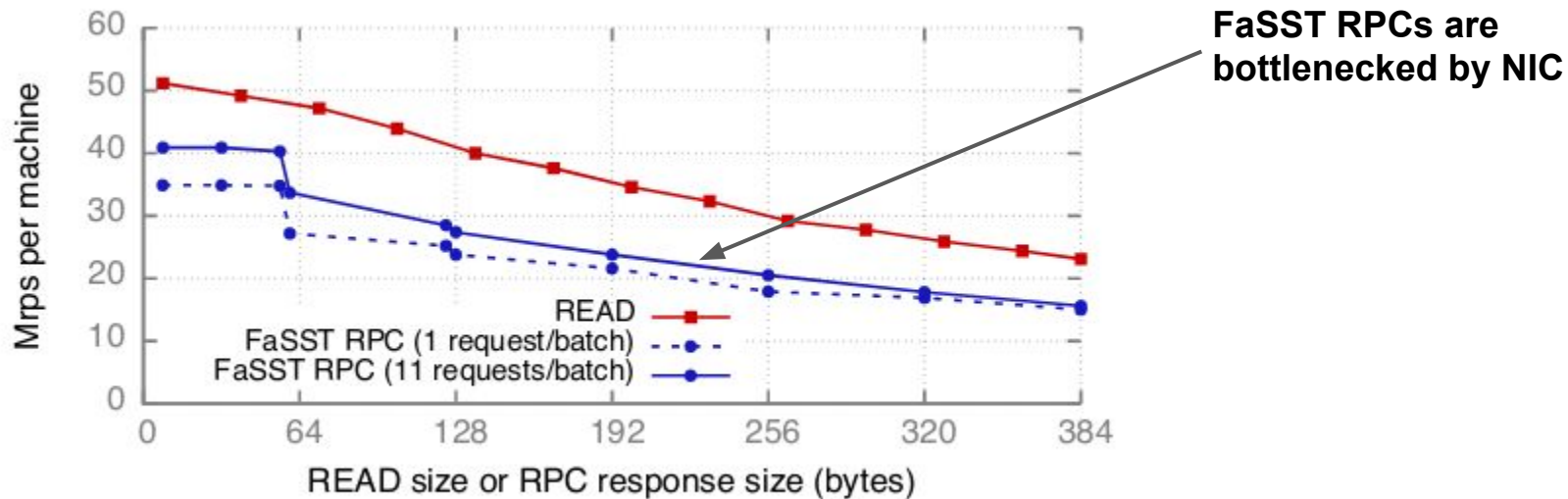
- Measure the raw/peak throughput
- 6 nodes in cluster for READs
 - On CX3, 8 cores so 48 QPs
 - On CIB, 14 cores so 84 QPs
 - Using 11 nodes gives lower throughput due to NIC cache misses
 - 1 READ for RDMA
- 11 nodes in cluster for RPCs
 - Using 6 nodes would restrict max non-coalesced batch size to 6
 - On CX3, 8 cores so 8 QPs
 - On CIB, 14 cores so 14 QPS
- Both READs and RPC have exclusive access to QPs in a small cluster
 - CPU is not the bottleneck
 - NIC is the bottleneck

Result- CX3 small cluster



(a) CX3 cluster (ConnectX-3 NIC)

Result- CIB small cluster



(b) CIB cluster (Connect-IB NIC)

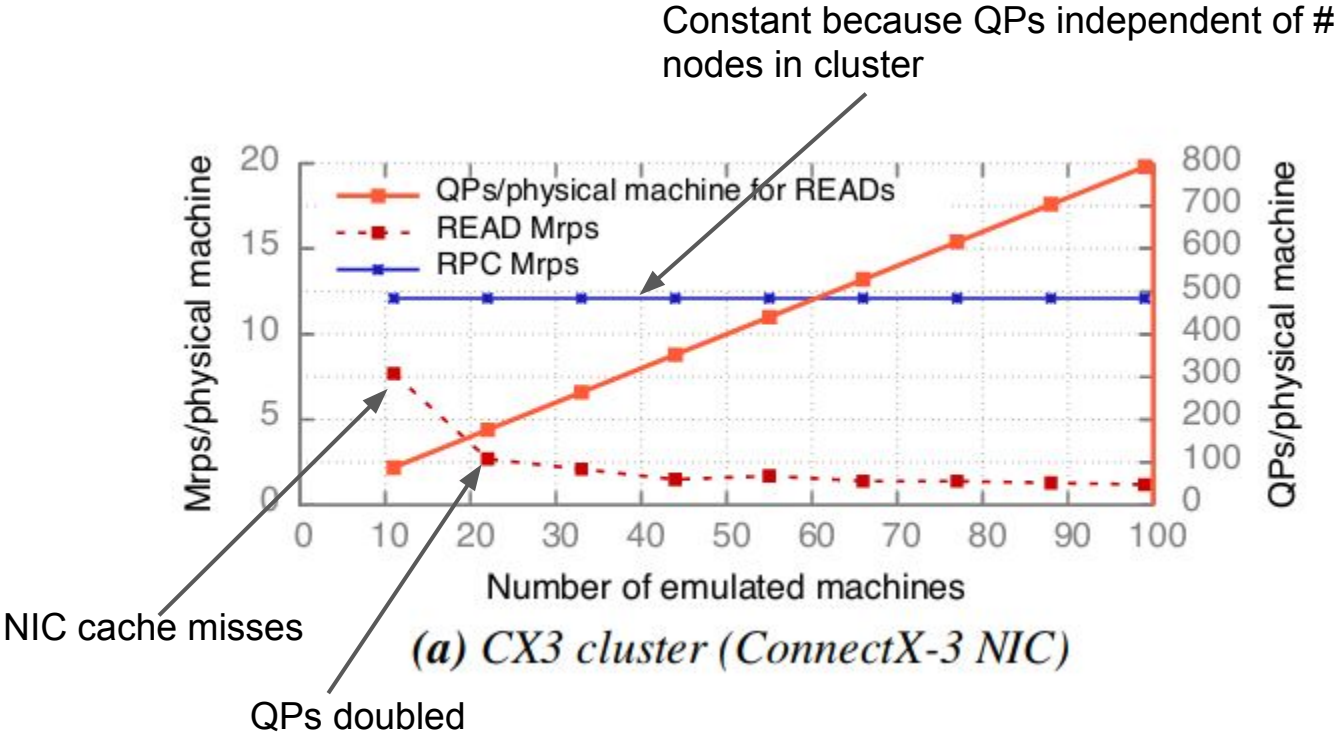
Effect of multiple reads vs RPCs

- RPCs provide higher throughput than using 2 or more READs
- Regardless of
 - Cluster size
 - Request size
 - Response size

Comparison on medium cluster

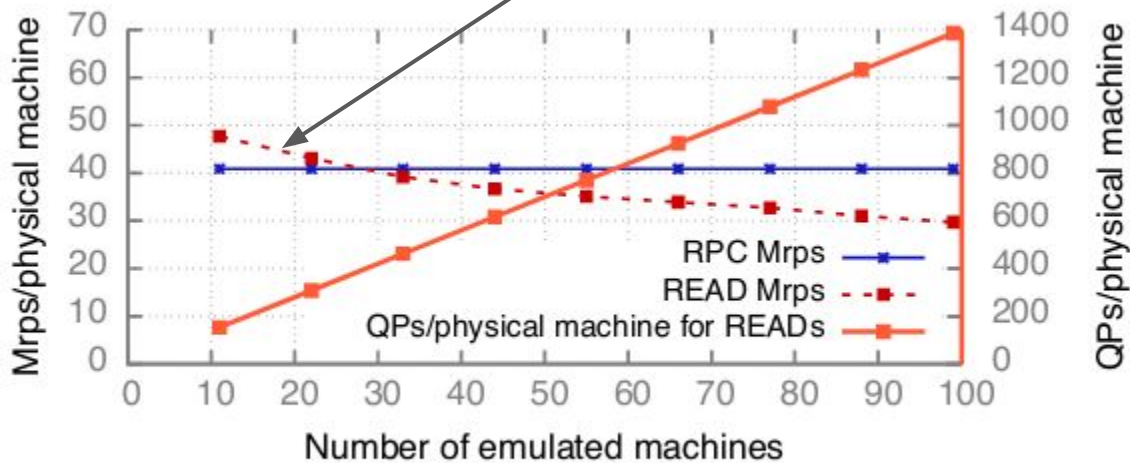
- Poor scalability for one-sided READs
- Emulate the effect of large cluster on CIB
 - Create more QPs on each machine
 - N physical nodes, emulate $N \cdot M$ nodes for varying M
 - For one-sided READs, $N \cdot M$ QPs
 - For RPC, QPs depends on # cores(14 in this case)
- FaSST RPCs performance is not degraded
 - QPs independent of cluster size

Result- CX3 medium cluster



Result- CIB medium cluster

More gradual decline as compared to CX3 due to larger NIC cache in CIB



(b) CIB cluster (Connect-IB NIC)

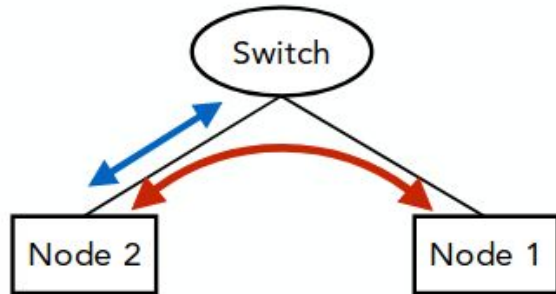
Shared QPs

- QPs shared between threads in one-sided RDMA
 - Fewer QPs so lesser NIC cache misses
 - CPU efficiency reduced
 - Lock handling required
 - Advantage of bypassing remote CPU is gone
- RPCs do not use shared QPs
 - Overall less CPU cycles required in a cluster setup

Local CPU cycles overhead offsets the advantage of bypassing the remote CPU in one-sided RDMA.

Reliability

**UD does not provide reliability.
But the link layer does!**

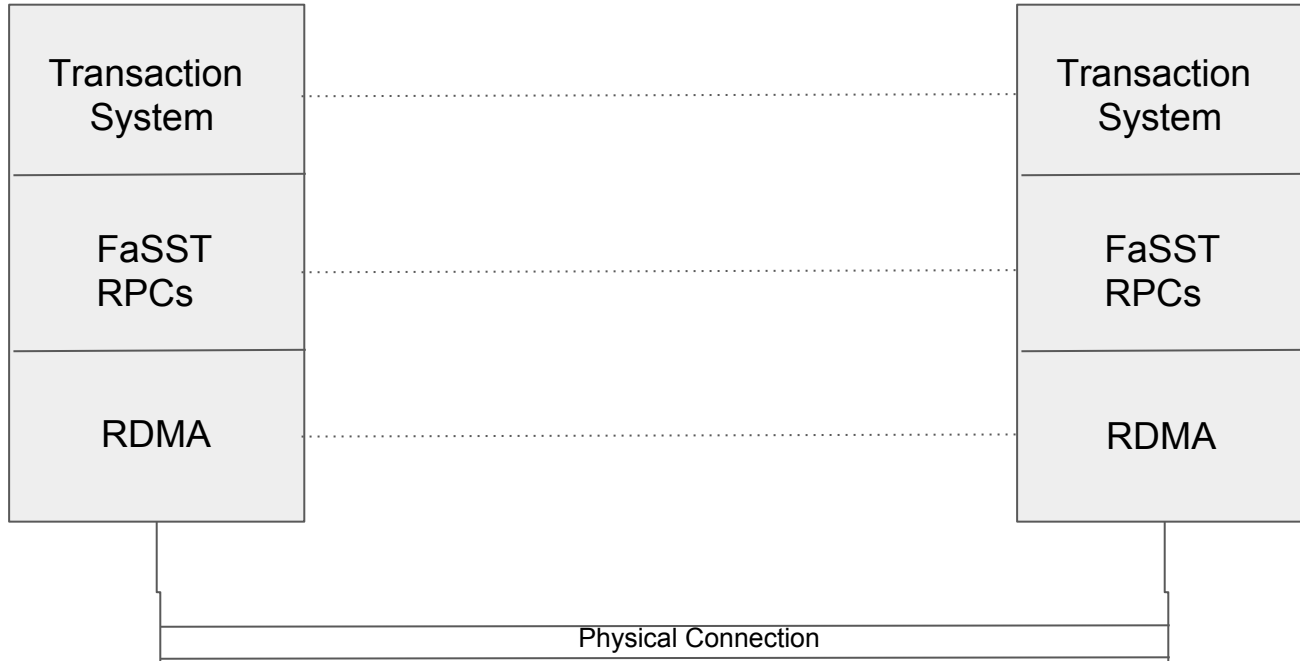


- No end-to-end reliability
- + Link layer flow control
- + Link layer retransmission

No packet loss in

- 69 nodes, 46 hours
- 100 trillion packets
- 50 PB transferred

Abstraction layers

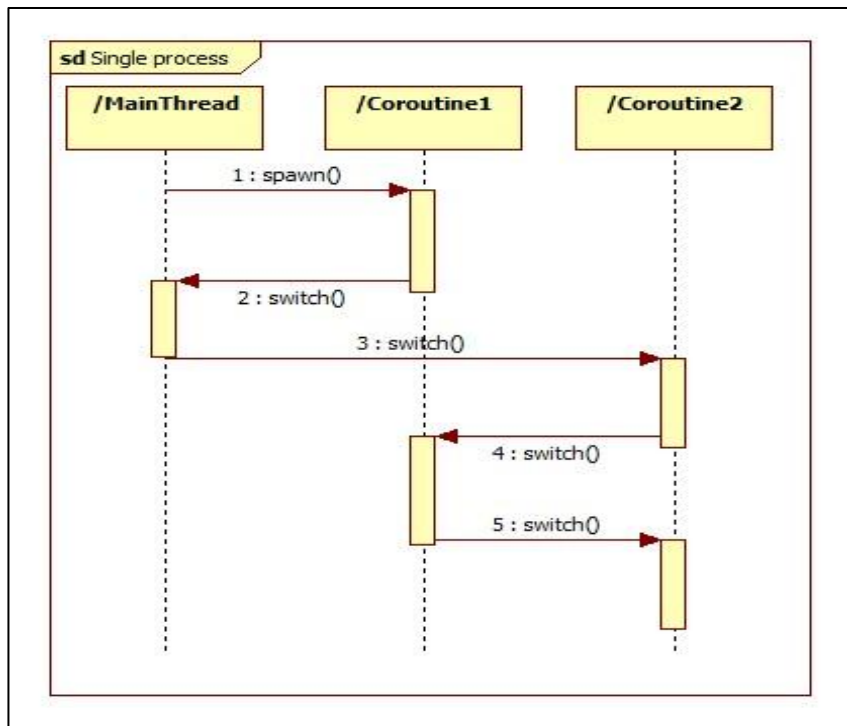


FaSST RPCs

FaSST RPCs

- Designed for transaction workload
- Small objects(~100 byte) and few tens of keys
- Integration with **coroutines** for network latency hiding(10 us)
 - ~20 coroutines are sufficient to hide network latency

Coroutines



- Blocking IO is not needed
- Cooperative/non-preemptive multitasking
- Yields after initiating network IO
- Master thread
 - One RPC endpoint per thread, shared among **Master coroutine** and **Worker coroutine**
- Switch between coroutines takes 13-20 ns

Why coroutines?

- With coroutines, the programmer and programming language determine when to switch coroutines.
- Tasks are cooperatively multitasked by pausing and resuming functions at set points.
- Pre-emption might not be in sync with the application, in case of normal threads.
- Less switching overhead

```
coroutine func {  
    yield Task1;  
    yield Task2;  
    yield Task3;  
}
```

```
int main() {  
    print func();  
    print func();  
    print func();  
}
```

Output - Task1,Task2,Task3

Optimizations

- Source - destination thread mapping
 - Restrict RPC communication between peer threads

Optimizations

- Source - destination thread mapping
 - Restrict RPC communication between peer threads
- Request batching
 - Reduces number of doorbell from b to 1
 - Allows RPC layer to coalesce messages sent to one machine
 - Reduces coroutine switching overhead(master yields only after receiving b responses)

Optimizations

- Source - destination thread mapping
 - Restrict RPC communication between peer threads
- Request batching
 - Reduces number of doorbell from b to 1
 - Allows RPC layer to coalesce messages sent to one machine
 - Reduces coroutine switching overhead(master yields only after receiving b responses)
- Response batching
 - Similar advantages as request batching

Optimizations

- Source - destination thread mapping
 - Restrict RPC communication between peer threads
- Request batching
 - Reduces number of doorbell from b to 1
 - Allows RPC layer to coalesce messages sent to one machine
 - Reduces coroutine switching overhead(master yields only after receiving b responses)
- Response batching
 - Similar advantages as request batching
- Cheap RECV posting
 - RECV requires creating descriptors in the RECV queue
 - Descriptor transfer from memory to NIC using DMA
 - DMA reads reduces CPU overheads

Detect packet loss

- Master counts the response for each worker to track the progress
- Master thread block worker, if it doesn't receive b responses before timeout
- Counter for worker doesn't change till timeout(1 second)
 - Packet loss
 - All worker threads can commit transaction before packet loss detection
- False positive for smaller timeout values
- Master kills the process in case of a packet loss

RPC Limitations

- MTU
 - 4096 bytes
 - Could be solved with segmentation in RPC layer
- Receive queue size
 - **One message per destination** to reduce the NIC cache thrashing
 - $N * t * c$ [**N** nodes, **t** threads/node, and **c** coroutines per thread]
 - Requires **t** queues of size $N * c * m$ [**m messages per destination**]
 - **t** queues of size $N * c$

Single-core RPC performance

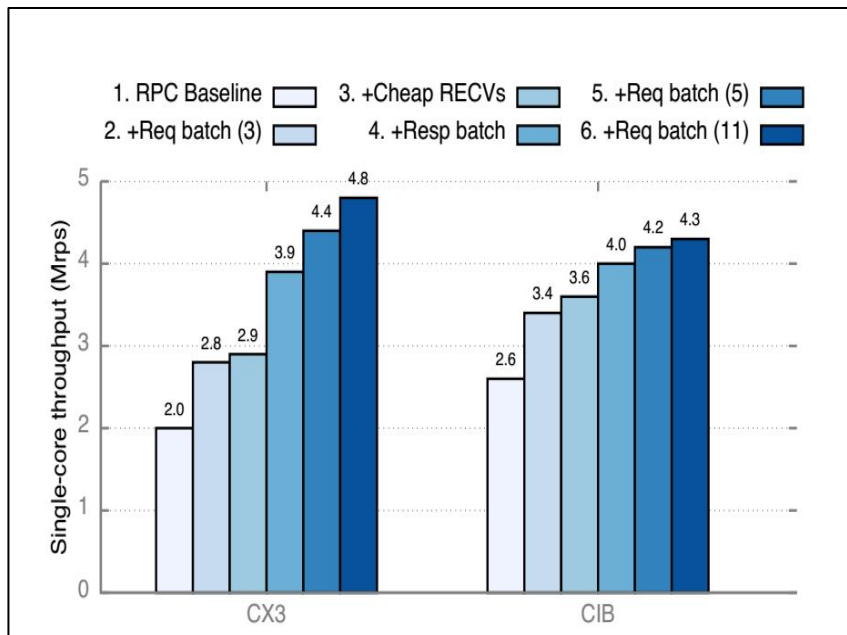
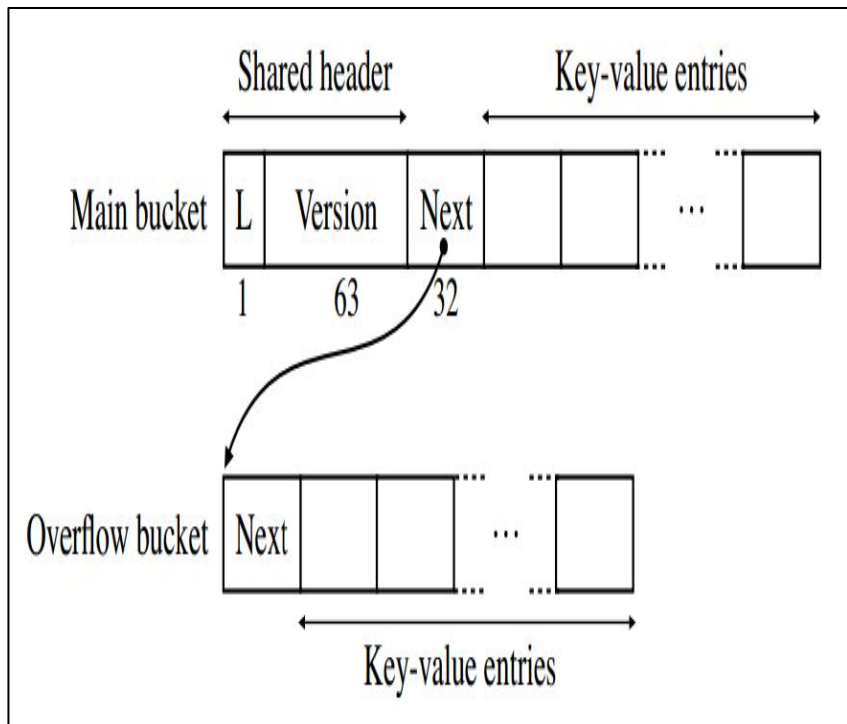


Figure: Per-core RPC throughput as optimizations 2–6 are added

- 2.0 Mrps for READs with QP shared between 3 or more threads
 - CIB baseline 2.6 Mrps
 - CIB maximum 4.3 Mrps : > 2x gain
- For 4.3 Mrps
 - 4.3 million SENDs for requests
 - 4.3 million SENDs for requests
 - 8.6 million for their RECVs
 - Total 17.2 million verbs per second
 - One-sided READs can achieve 2 million verbs per second

Transactions

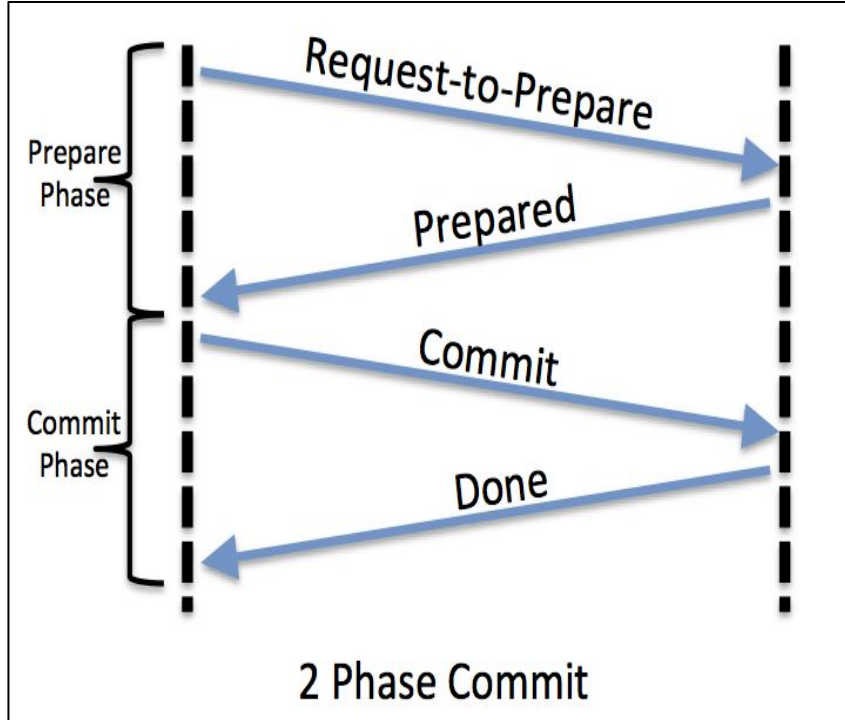
Bucket layout



- 8 byte keys
- Upto 4060 byte values
- 8 byte headers
 - Concurrency control
 - Ordering commit log records during recovery
 - Several keys can map to same header

Figure: Layout of main and overflow buckets in MICA based hash table

Two-phase Commit



- Prepare phase
 - Each slave sends DONE to master
 - Master sends READY? to each slave
- Commit phase
 - Master sends COMMIT to all slaves
 - Each slave sends ACK to master

Optimistic Concurrency Control Phases

- **Begin**
 - Record a timestamp marking the transaction's beginning.
- **Modify**
 - Read database values, and tentatively write changes.
- **Validate**
 - Check whether other transactions have modified data that this transaction has used.
- **Commit/Rollback**
 - If there is no conflict, make all changes take effect. If there is a conflict, resolve it, typically by aborting the transaction, although other resolution schemes are possible.

Coordinator log based two-phase commit

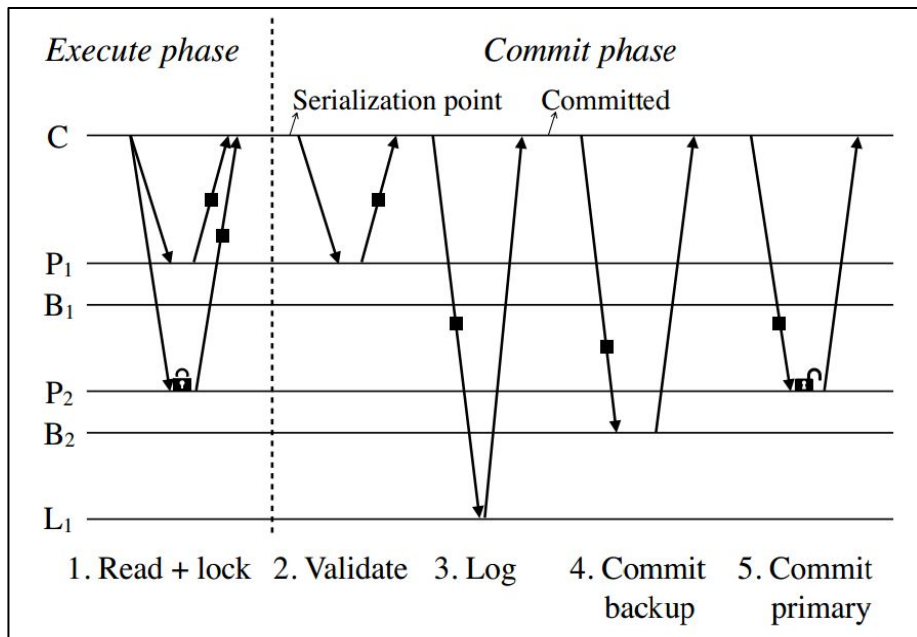


Figure: FaSST's transaction protocol with tolerance for one node failure. **P1** and **P2** are **primaries** and **B1** and **B2** are their backups. **C** is the **transaction coordinator**, whose **log replica** is **L1**. The solid boxes denote messages containing application level objects. The transaction **reads** one key from **P1** and **P2**, and **updates** the key on **P2**.

Handle failure and packet loss

- FaSST provides serializability and durability, but not high availability
- Machine failure recovery mechanism(*have not implemented*)
 - Leases, cluster membership reconfiguration, log replay and log replication
- Convert packet loss to machine failure
 - Kill the FaSST process
- No packet loss for 50 PB of data
 - Rare event
 - Each failure is 5x50 ms of down time -> 99.999% availability.

Implementation

- Handler for **get**, **lock**, **put**, and **delete**
- User registers for table and respective handlers
- RPC request type decides which table to refer
- Exclusive data store partition per thread
 - Not scalable in clustered setup, require large RECV queue size
- Transaction APIs
 - **AddToReadSet(K, *V)** and **AddToWriteSet(K, *V, mode)**
 - Mode - insert, update, delete.
 - **Execute()** - All requests in one go, to support doorbell batching.
 - **Abort()** - If the key is locked.
 - **Commit()** - Runs the complete protocol i.e validation, logging and commit.

Evaluation

Workloads

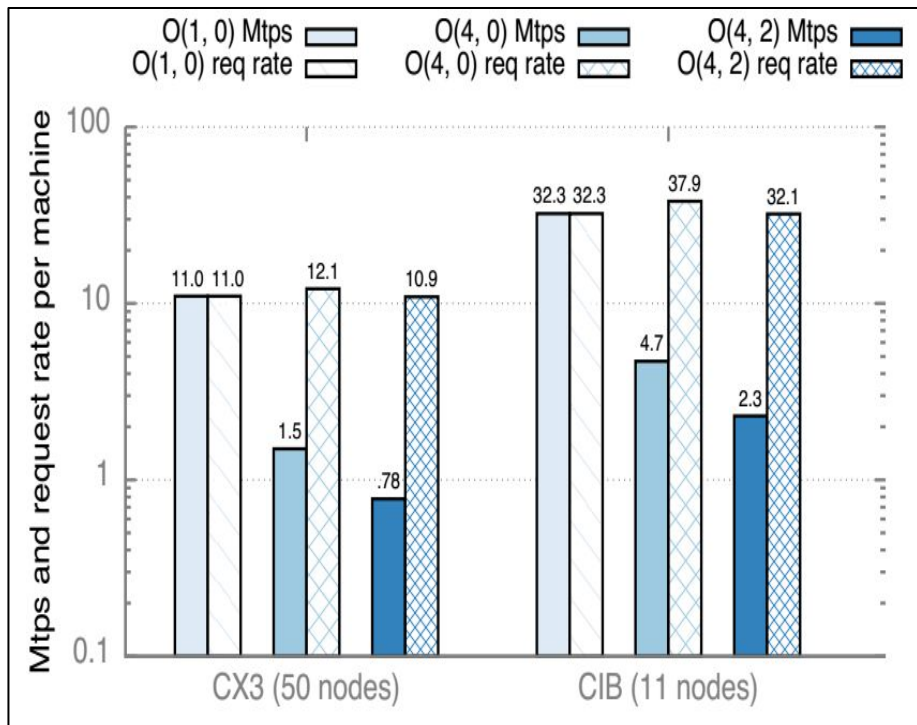
- Object store
 - Read-mostly OLTP benchmark
 - Effect due to multi-key transactions and write-intensiveness
- TATP
 - Simulates a telecommunication provider's database
 - 70% transactions read 1 key
 - 10% transactions read 1-4 key
 - 20% transactions modify key
- SmallBank
 - Simulates bank account transactions
 - 85% transaction update a key

3-way logging and replication

Setup comparison

	Nodes	NICs	CPUs(core used, GHz)
FaSST	50	1	1x E5-2450 (8, 2.1 GHz)
FaRM	90	2	2x E5-2650 (16, 2.0 GHz)
DrTM+R	6	1	1x E5-2450-v3 (8, 2.3 GHz)

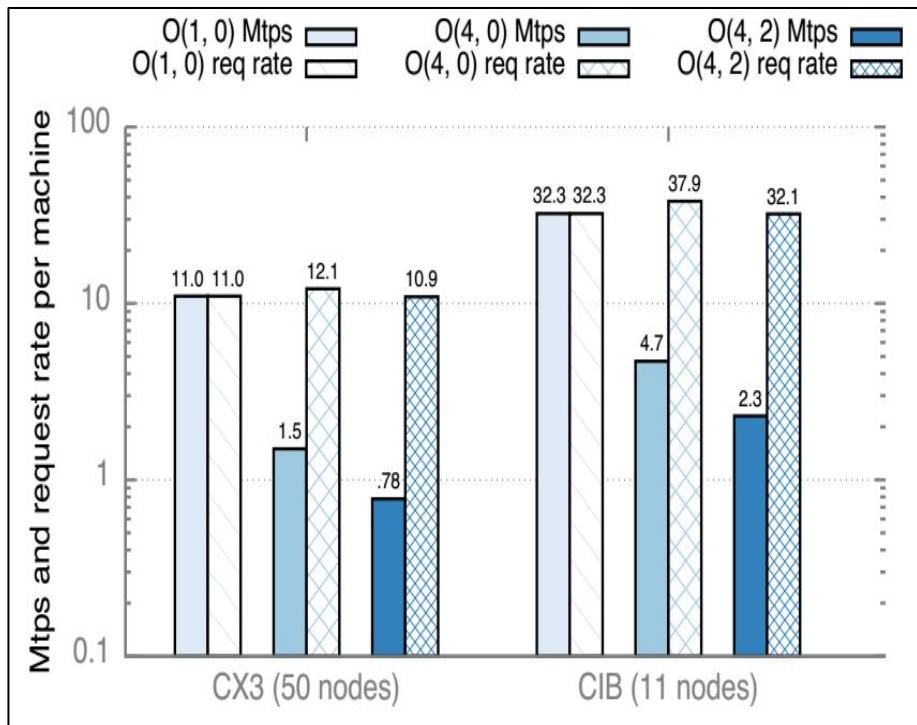
Single-key vs multi-key transactions



- 8 byte keys and 40 byte values
- 1M keys per thread in cluster.

- $O(r, w)$ - Read r key and update w keys
- $O(1, 0)$ - single-key read-only transaction
- $O(4, 0)$ - multi-key read-only transaction
- $O(4, 2)$ - multi-key read-write transaction

Single-key read-only transactions

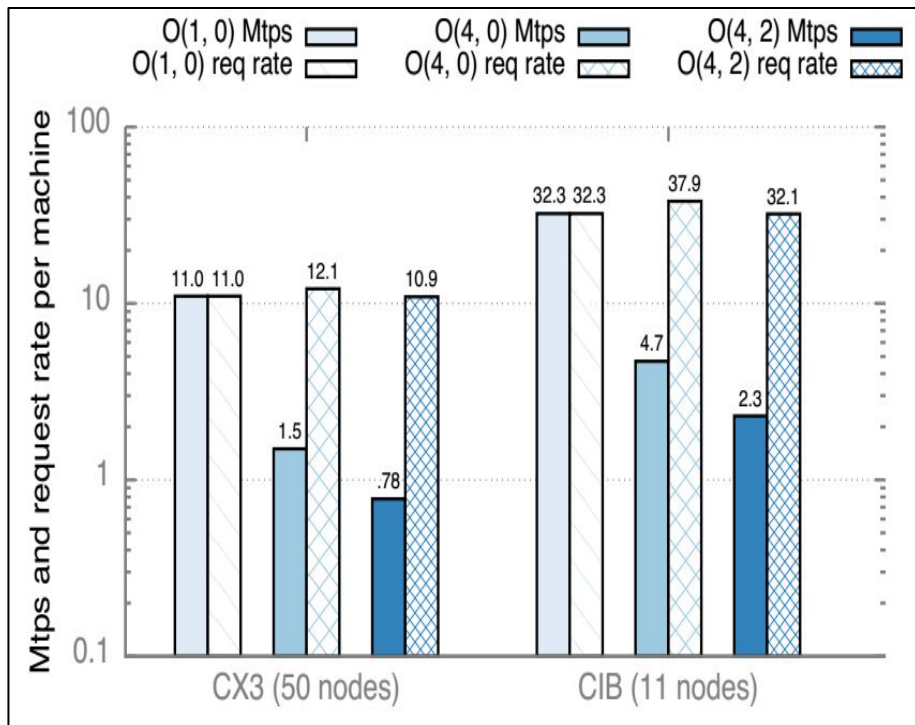


- CX3 - Bottlenecked by NIC@11 Mrps
- CIB - CPU bottleneck
 - No doorbell batching for requests

Comparison

- FaRM **90** machine cluster and FaSST **50**
- Suited to FaRM's design goal to bypass remote CPU
 - Local CPU is the bottleneck
- 1.25x higher throughput per machine with less resources per machine

Multi-key transactions



- O(4,0) larger transactions
 - Reason for throughput decrease
- Both CX3 and CIB are bottlenecked by NICs

- O(4,2) larger transactions
- CPU bottleneck because of inserts into the replicas on CIB

Comparison for read-intensive workload

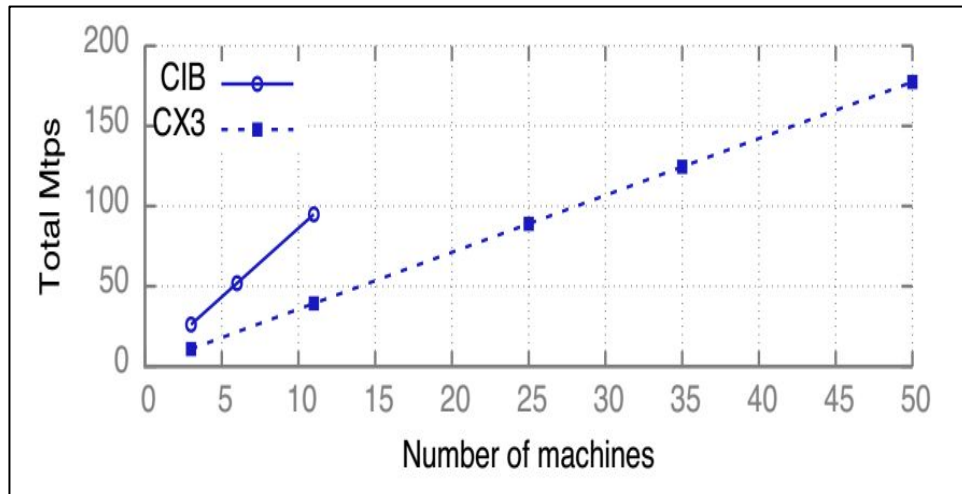


Figure: TATP Throughput

- 70% single key read, 10% 1-4 key read and 20% key modify
- Scales linearly
- FaSST performs 87% better than FaRM on 50 nodes cluster

Comparison for write-intensive workload

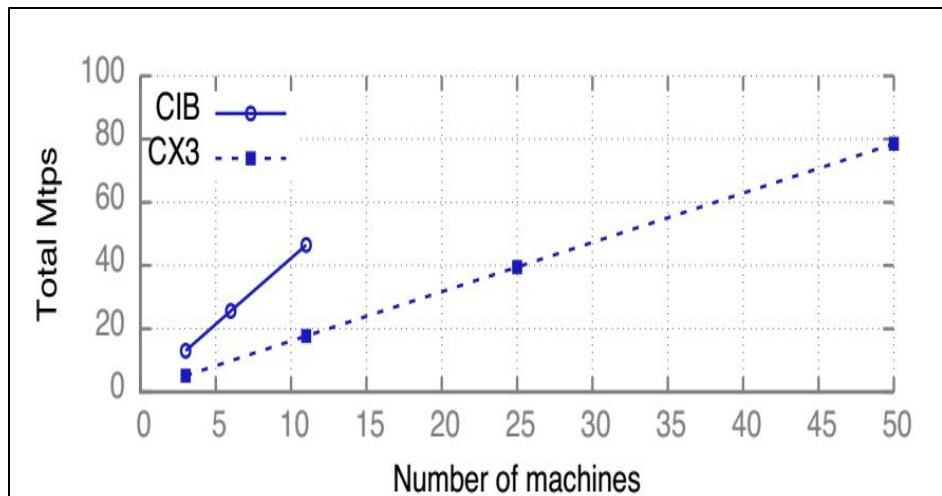
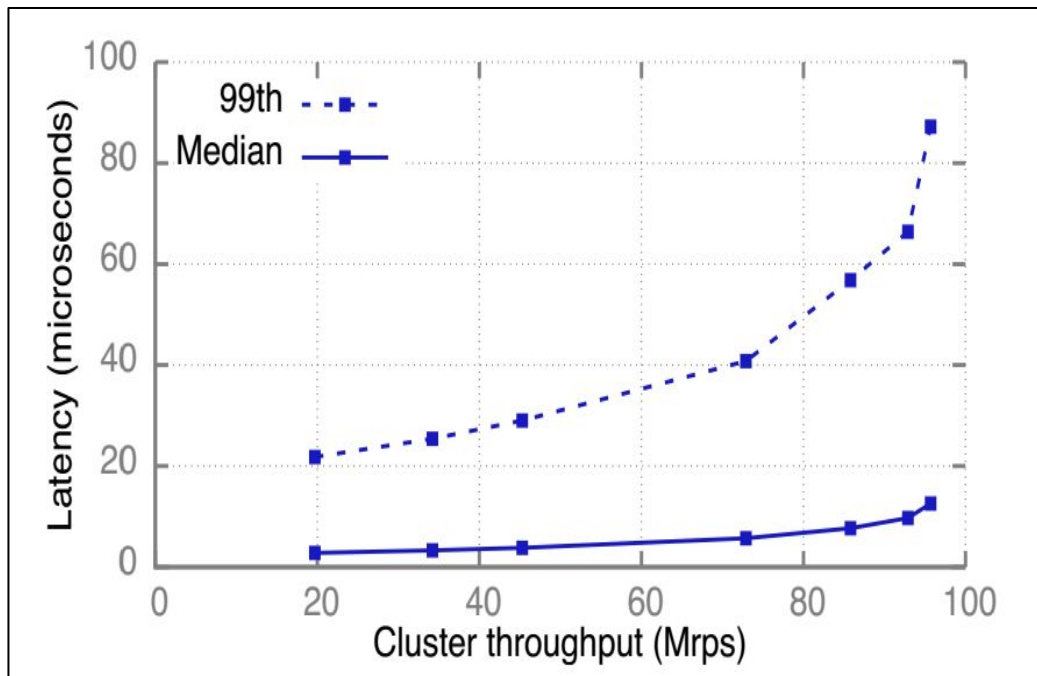


Figure: SmallBank throughput

- 1 Lakh bank accounts per thread
- 4% of total accounts accessed by 90% of transactions
- Scales linearly in this case too

- FaSST outperforms DrTM+R by 1.68x on CX3 and 4.5x on CIB
- DrTM+R is slow because
 - 4 one way operations as compared to 2 in FaSST for write operation
 - ATOMICs are expensive on CX3
 - May be affected by NIC cache misses as no QP sharing in DrTM

Latency



- TATP workload
- Latency for committed transactions
- 14 threads per machine
 - 1-19 coroutines per thread
- One worker per thread
 - 19.7 Mrps
 - 2.8 us median latency
 - 21.8 us 99th percentile latency
- 19 workers per thread
 - 95.7 Mrps
 - 12.6 us median latency
 - 87.2 us 99th percentile latency

Future trends and their effects on FaSST

Scalable one-sided RDMA

- Dynamically Connected Transport
 - 3 messages for QP change, large overhead for large fanout workload
 - NIC cache misses due to frequent QP change
- Portals: Scalable one-sided RDMA using connectionless design
 - Multiple round trip to access datastore
 - Scalable one-sided WRITE might outperform FaSST

- Best design will likely be hybrid of RPCs and remote bypass
 - RPCs used for accessing data structures
 - Scalable one-sided WRITES for logging and replication.

More queue pairs

- CIB(new) can cache larger QPs as compared to CX3(old)
- QPs are increasing in newer NICs
- #Cores are also increasing in newer CPUs
- Sharing QP is not a good idea

Supports FaSST's datagram based design

Advanced one-sided RDMA

- Even if NIC can support multi-address atomic operation and B-Tree traversals
 - NIC to Memory path is costly
 - CPU onload is better in such cases and not NIC offload

FaSST is expected to work well in these scenarios.

Conclusion

Transactions with one-sided RDMA are:

- **Slow:** Data access requires multiple round trips
- **Non-scalable:** Connected transports
- **Complex:** Redesign data stores

Transactions with two-sided datagram RPCs are:

- **Fast:** One round trip
- **Scalable:** Datagram transport + link layer reliability
- **Simple:** Re-use existing data stores

FaSST outperforms by 1.68x-1.87x, with fewer resources and without workload assumptions.

Thank You