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

Anuj Kalia (CMU)
Michael Kaminsky (Intel Labs)
David Andersen (CMU)
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**
In-memory distributed transactions

Distributed ACID transactions can be fast in datacenters

FaRM [SOSP 15, NSDI 14], DrTM [SOSP 15, EuroSys 15], RSI [VLDB 16]

Enablers:

1. Cheap DRAM, NVRAM: No slow components on critical path
2. Fast networks: Low communication overhead
How to access remote data structures?

**Transaction environment**

**Existing systems**

- **Method**: One-sided READs
- **Round trips**: \( \geq 2 \)

**FaSST**

- **Method**: Two-sided RPCs
- **Round trips**: 1

- **Node 1**
  - READ (pointer)
  - READ (value)

- **Node 2**

- **RPC request**

- **RPC response**
RPC v/s READs microbenchmark

FaSST RPCs make transactions faster

FaRM [SOSP 15, Fig 2]
(2x ConnectX-3 NICs)

CPU-limited

FaSST
(1x Connect-IB NIC)
NIC-limited
# Reasons for slow RPCS

<table>
<thead>
<tr>
<th>Method</th>
<th>One-sided READs</th>
<th>Two-sided RPCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round trips</td>
<td>$\geq 2$</td>
<td>1</td>
</tr>
<tr>
<td>Scalable transport</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Effect: NIC cache misses</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lock-free I/O</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Effect: Low per-thread tput</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Existing systems**

**FaSST**
One-sided RDMA does not scale

READs & WRITEs must use a connected transport layer

One-sided systems

Problem: Cache overflow

Node 1

Thread

Thread

Node 1

Node 2

Node 3

Node N

READ (Reliable Connected)

RPC req WRITE (Reliable Connected)

RPC resp WRITE (Reliable Connected)

NIC cache
CPU overhead of connection sharing

Problem: Cache overflow
Problem: Connection sharing

Local overhead of remote bypass = 5x
Connectionless transport scales

But it supports only two-sided (SEND/RECV) operations

READs don’t use fewer CPU cycles than RPCs!
Local overhead offsets remote gains

FaSST RPCs make transactions *scalable*

![Diagram showing network nodes and connections](image-url)

- NIC cache
- Thread
- Node 1
- Node 3
- Node N

<table>
<thead>
<tr>
<th>Req rate/thread (M/s)</th>
<th>READs (sharing)</th>
<th>FaSST RPCs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.1</td>
<td>3.0</td>
</tr>
</tbody>
</table>

FaSST RPCs make transactions more scalable because they use fewer CPU cycles compared to READs. Local overhead offsets remote gains.
FaSST RPCs make transactions **Simpler**

Remote bypassing designs are complex

- Redesign and rewrite data stores
- Hash table [FaRM-KV, *NSDI 14*], B-Tree [Cell, *ATC 15*]

RPC-based designs are simple

- Reuse existing data stores
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

Handle packet loss similar to machine failure: See paper
Performance comparison

**Tput/machine (M/s)**

<table>
<thead>
<tr>
<th>Nodes</th>
<th>NICs</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>FaRM</td>
<td>50</td>
<td>2x ConnectX-3</td>
</tr>
<tr>
<td>DrTM+R</td>
<td>6</td>
<td>1x ConnectX-3</td>
</tr>
<tr>
<td>FaSST</td>
<td>50</td>
<td>1x ConnectX-3</td>
</tr>
</tbody>
</table>

**TATP benchmark**
(80% rdonly txns)

- FaRM: 1.9 M/s
- FaSST: 3.6 M/s

**SmallBank benchmark**
(85% rw txns)

- DrTM+R: 0.9 M/s
- FaSST: 1.6 M/s

**vs FaRM:** FaSST uses 50% fewer h/w resources

**vs DrTM+R:** FaSST makes no data locality assumptions
Transactions with one-sided RDMA are:

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

Transactions with two-sided datagram RPCs are:

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

**Code**: [https://github.com/efficient/fasst](https://github.com/efficient/fasst)