FASST: FAST, SCALABLE, AND SIMPLE DISTRIBUTED TRANSACTIONS

Shivaram Venkataraman
Fall 2018
MOTIVATION

RDMA is great! We can build fast distributed stores!

Existing systems all use 1-sided RDMA
- Need for multiple round trips for B-Trees etc.
- Need to maintain connection state (queue pairs)

Approach: Design RPC layer that is fast, simple, scalable
ONE-SIDED VS TWO-SIDED

Node 1

One-sided (READ)

Node 2

Two-sided (SEND)

NIC

CPU

RECV

DRAM
### COMPARING RDMA MODES

<table>
<thead>
<tr>
<th></th>
<th>SEND/RECV</th>
<th>WRITE</th>
<th>READ/ATOMIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>UC</td>
<td>✔</td>
<td>✔</td>
<td>X</td>
</tr>
<tr>
<td>UD</td>
<td>✔</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Table 1:** Verbs supported by each transport type. RC, UC, and UD stand for Reliable Connected, Unreliable Connected, and Unreliable Datagram, respectively.
PAPER CONTRIBUTIONS

1. Design RPC using two-sided unreliable datagram verbs

2. Support parallel RPCs using co-routines

3. Optimizations for batching

4. Detect / Handle packet loss ?
NEED FOR DATAGRAM RPCS

How to do index operations?
   FaRM: Inline values with keys
   DrTM: Replicate index

Queue pair scaling
   Connection state per thread to all recipients
   Optimizations like sharing queue pairs (affect performance)
   Datagram transport require no state!
DATAGRAM RPCS VS ONE-SIDED

(a) CX3 cluster (ConnectX-3 NIC)

Real cluster of 6 nodes

Emulated cluster

Graph showing performance comparison between different RPC methods.
**FASST RPCS**

**Coroutines**
- RDMA latency ~10us
- Use coroutines to yield while waiting for response
- Small number (~20) coroutines per thread

**Master/worker**
- Master co-routine handles request from remote machines
- Workers run application logic and issue RPC requests
Request Batching
- Each request has to ring NIC “Doorbell” from CPU
- Coalesce multiple messages (e.g., multi-key transaction)
- Invoke coroutine once per batch
- Batching is *opportunisitic*

Cheap RECV posting
- Need to limit size of RECV queue
- Required modifying *NIC driver*
RELIABILITY

- 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
Handling packet loss

Use timeout to check if coroutine got reply
On timeout, kill the FaSST process on the machine!
Timeouts can be large – don’t affect other threads
Application-level recovery (second talk)

Pros/cons of this approach?
LIMITATIONS

RPC messages smaller than MTU (4KB)

Each co-routine issues one message per destination per batch
Why? Keep RECV queues small
FASST TRANSACTIONS

Execute phase

Commit phase

C
P_1
B_1
P_2
B_2
L_1

1. Read + lock
2. Validate
3. Log
4. Commit backup
5. Commit primary

Serialization point
Committed
Applications create read sets and write set

AddToReadSet(K, *V) and AddToWriteSet(K, *V, mode)

Lazily evaluated (not run until Execute is called)

Allows batching

Applications can call Execute multiple times!

Transaction status

Commit() / Abort() based on transaction result
SUMMARY

One-sided RDMA read vs two-sided RDMA RPC

RPCs: useful building block
Need to handle link reliability

More debate:
“Deconstructing RDMA-enabled Distributed Transactions: Hybrid is Better!”