FaRM: Fast Remote Memory

Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro, **Microsoft Research**

Distributed Stores

Distributed Stores

- Became popular in last 5-10 years with decreasing cost of DRAM:
  - With 128GB of memory per machine, 32 machines can store 4TB of data in RAM
  - Frequently, a modest sized cluster can fit the entire working set of an application in memory

Performance: Get/Put

- Made up of several factors:
    - Latency to identify where key is stored
    - Network request latency
    - Time needed to get key from host
- Multiplied by additional protocol overhead —> e.g., two phase commit

Network Performance

- As a vast overgeneralization, datacenter networks do not behave:
  - Large variance in terms of flows (elephants vs. mice), synchronization of flows, etc.
- Additionally, short lived connections don’t perform great under TCP:
  - Need to pay connection setup time, slow start

FaRM Thesis:

For max performance, don’t use TCP/IP, use RDMA

What is RDMA?

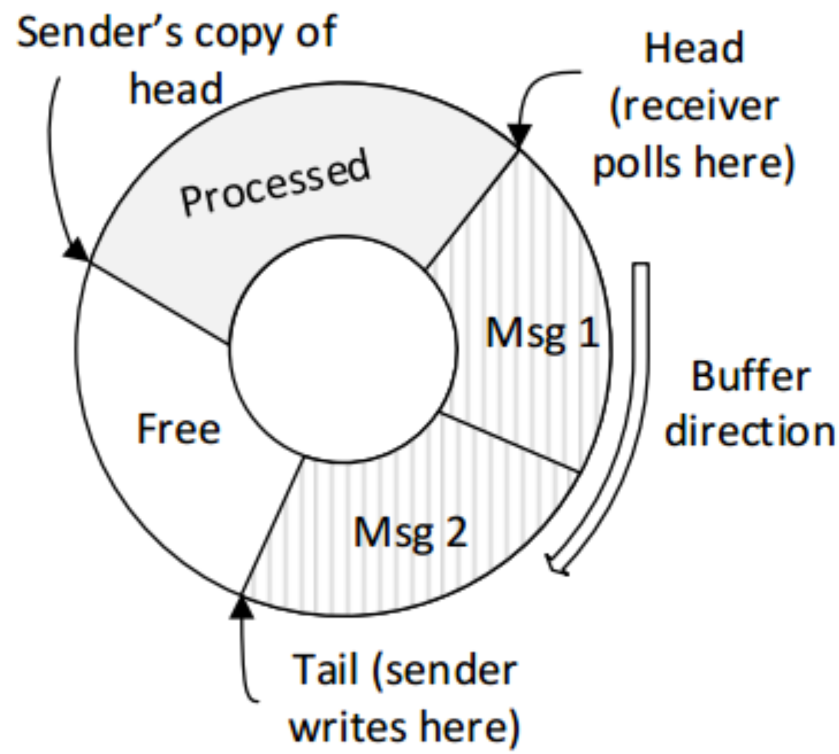
- RDMA is networking abstraction that provides direct access to memory on a remote machine
- Just like traditional DMA, RDMA has lower overhead:
  - Memory access on remote node is a DMA from NIC; processor not involved
  - Bypasses traditional TCP/IP stack

So, just use RDMA and we're done, right?

Fast message passing

Circular message queue is manipulated via RDMA:

1. Sender tracks head ptr
2. Sender writes at tail ptr
3. Sender increases tail ptr
4. Receiver lazily updates sender's head ptr



Beyond circular buffers

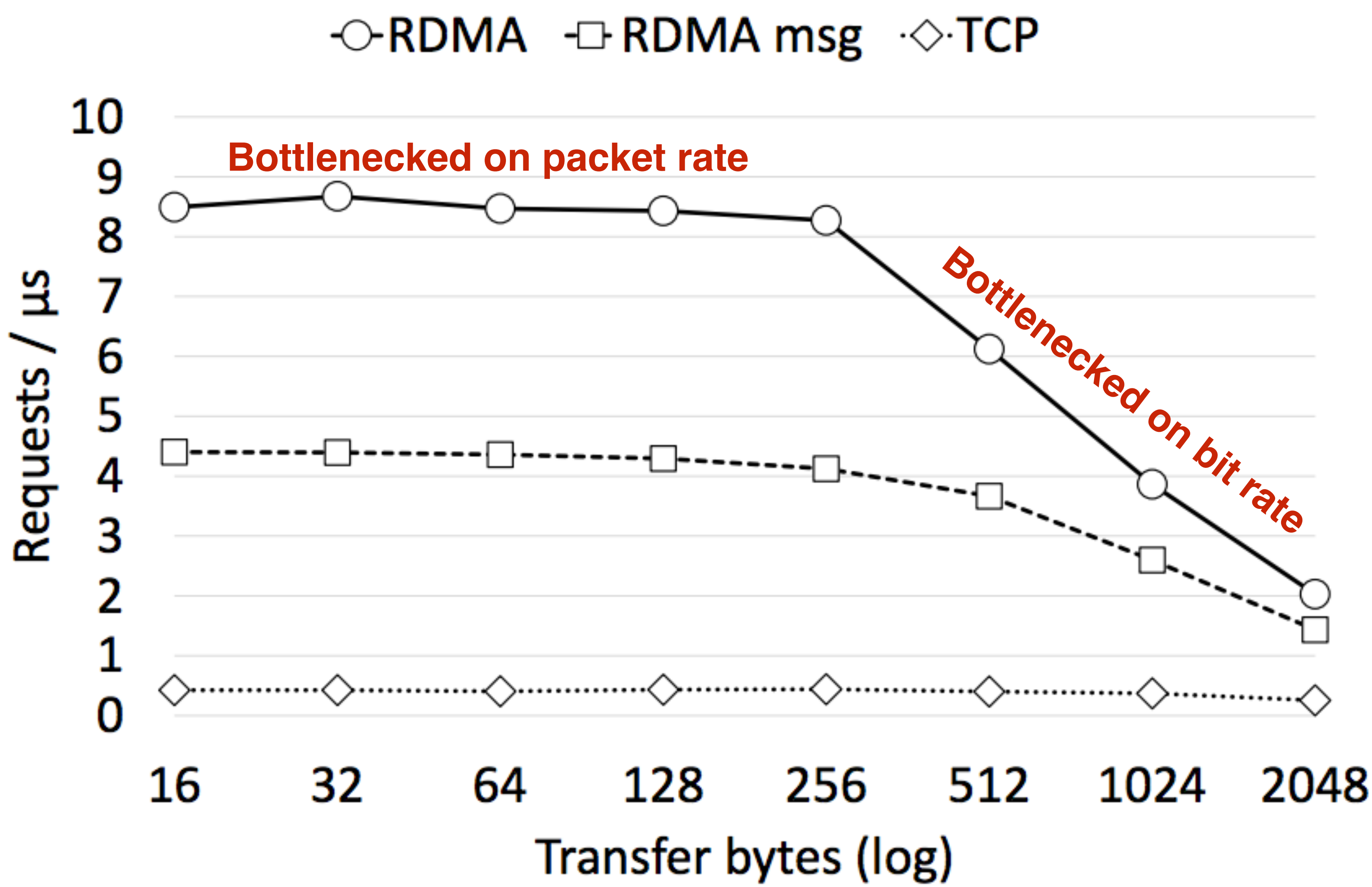
- Three additional hacks:

  1. NIC page table is too small to store large page table; instead use 2GB überpages

  2. NIC can't cache message queues; improve by reducing the number of message queues by *tq* <— *t* is threads per machine, *q* is a "NUMA-aware" factor

  3. Interrupts increase RDMA latency by 4x; pin response threads to hardware threads and poll

Raw Message Perf



Disappointing result:

RDMA still 23x slower than local memory

Actual (?) FaRM Thesis:

Locality is priceless,

for everything else, there is FaRM

FaRM API

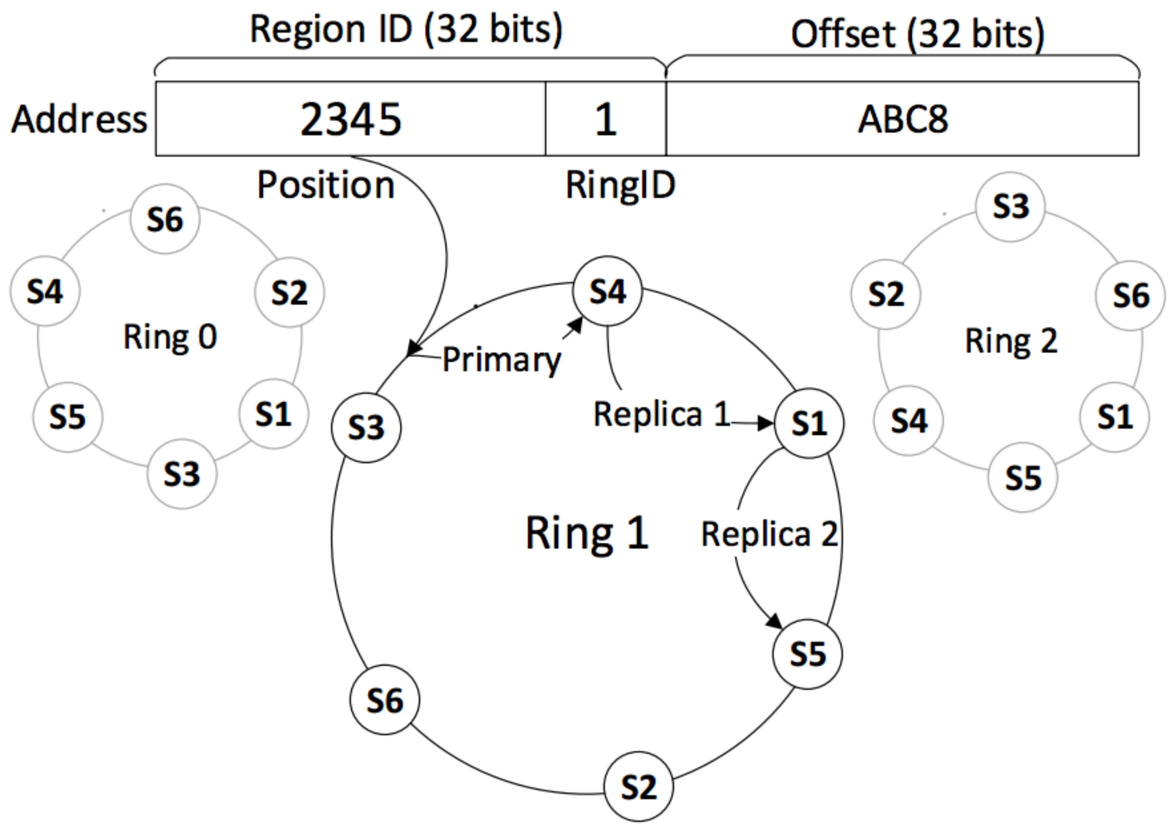
```
Tx* txCreate();
void txAlloc(Tx *t, int size, Addr a, Cont *c);
void txFree(Tx *t, Addr a, Cont *c);
void txRead(Tx *t, Addr a, int size, Cont *c);
void txWrite(Tx *t, ObjBuf *old, ObjBuf *new);
void txCommit(Tx *t, Cont *c);

Lf* lockFreeStart();
void lockFreeRead(Lf* op,Addr a,int size,Cont *c);
void lockFreeEnd(Lf *op);
Incarnation objGetIncarnation(ObjBuf *o);
void objIncrementIncarnation(ObjBuf *o);

void msgRegisterHandler(MsgId i, Cont *c);
void msgSend(Addr a, MsgId i, Msg *m, Cont *c);
```

- Global address space w/ opaque pointers
- Lock-free reads are serializable w/ transactions

Distributed Memory Management

- Objects are stored in 2GB regions, distributed across cluster
    - Top 32 bits of 64 bit address point to the memory region, low bits are offset
- Regions are located using a consistent hashing scheme
    - If object is remote, request capability from owner
    - Capability + offset + obj size —> RDMA request

Consistent Hashing Scheme



- Scheme has several rings; hash function per ring
- Hash IP address to get ring position

Memory Allocation

- Three level allocation scheme:
  - Region allocator —> cluster wide
  - Block allocator —> per machine
  - Slab allocator —> per thread
- Slab allocator groups objects into blocks by size; allocation sizes are fixed into 256 levels <1MB
- Allocator allows users to provide locality hints

Transactions vs. Lock-free operations in FaRM

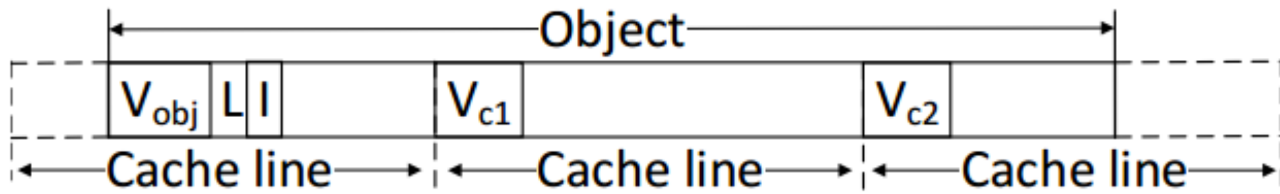
FaRM Transactions

- At high level, fairly vanilla 2PC transactions
- However, two optimizations:
  - RDMA!
  - Single machine transactions

Single Machine Txns

- Why do we need 2PC? Data is shared across machines.
- If all data needed to run a transaction is located on a single machine, we can run the transaction on the primary node
    - Eliminates prepare and validate phases of 2PC
    - However, data is replicated —> must ensure primary and replicas are same for all data.

Lock Free Reads in FaRM



- Uses a simple versioning scheme:
    - Version is written in object header and in each cache line
- If all versions match, and header is unlocked, we can make the read
- Else, retry after random back off

Lock Free Reads: Nifty Low Level Asides

- Object header is locked via cmp&swp during transaction prepare phase: this lock is visible to the lock-free read
- DMA is cache coherent on x86
- Prevent reads of freed objects by checking that incarnation value matches expected
- Don't store full version in cache line; store low bits and timeout reads that complete slowly
- …

FaRMing:

FaRM in action

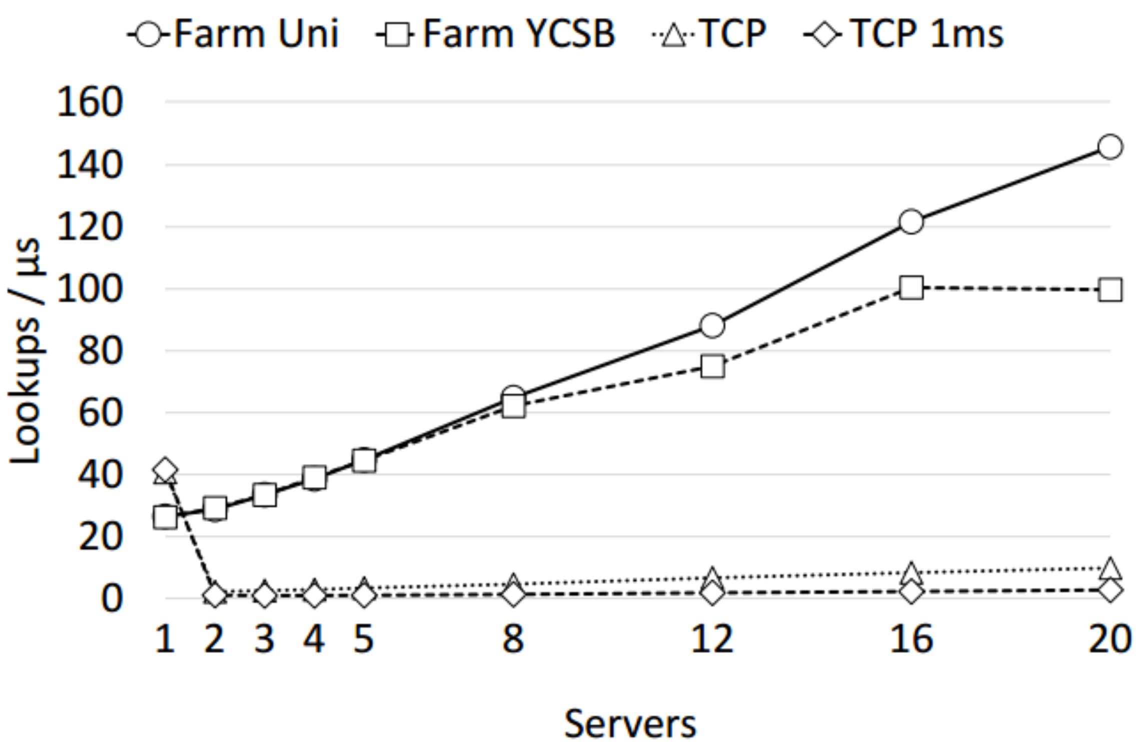
Two evaluations

- Isolated cluster of 20 machines, 40 Gbps RoCE
- KV Store
    - Compare vs. “something like” MemC3

    1. Uniform distribution of key accesses
    2. YCSB: “Real world” NoSQL benchmark suite
- Tao

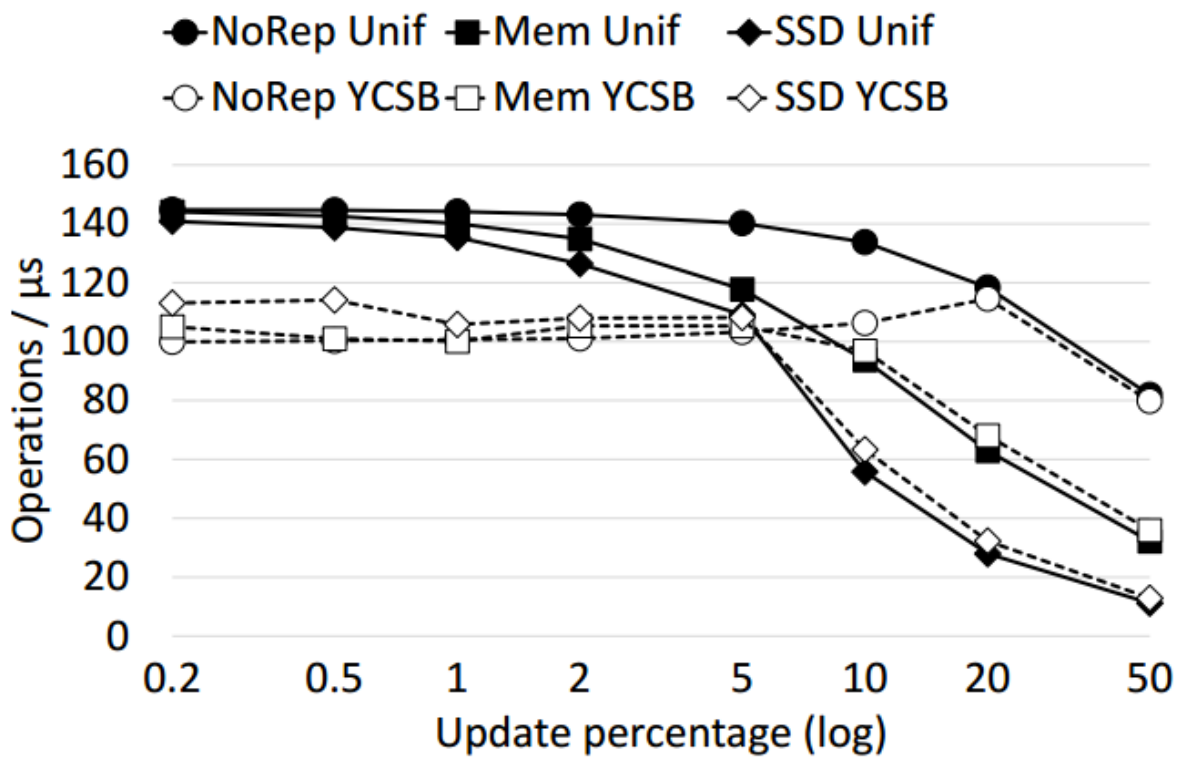
    1. Benchmark on Facebook LinkBench vs. reported Tao numbers

Get KV Store



- FaRM is approx 1.5x worse than baseline on a single machine
- Plateau at 16 nodes is caused by key skew

Put KV Store



- Higher overhead logging shifts perf knee
- Perf knee seems to imply where logging overhead is more significant than key skew?

Tao Evaluation

- Tao is 99.8% reads
- Implemented subset of Tao
- Throughput is 10x better than reported numbers for Tao
- Latency is 40x lower
- Each operation requires ~1 RDMA read

In summary…

What is FaRM?

- A “philosophy”:
  - Distributed systems work best when nodes don’t need to talk, but when they do talk, make it fast
- With lots of nifty engineering:
  - Make it possible to do lock-free consistent reads
  - Restructure your algorithms to avoid remote accesses
  - Etc.