The future of data centers

What do **you predict** does **the future** of data centers look like?

The future of data centers

Trends until now:

- **Horizontal scaling**
  Single threads are too slow. Moore's law doesn't apply.

We predict that:

- **Main memory will keep getting cheaper**
  The entire application already fits into the main memory.

  – 100GB to 1TB per server
  – 10 to 100TB per cluster

- **TCP/IP stack will then be the next bottleneck**
  FaRM suggests RDMA as an alternative.

The future of data centers

Trends until now:

- **Horizontal scaling**
  Single threads are too slow. Moore's law doesn't apply.

We predict that:

- **Main memory will keep getting cheaper**
  The entire application already fits into the main memory.

  – 100GB to 1TB per server
  – 10 to 100TB per cluster

- **TCP/IP stack will then be the next bottleneck**
  FaRM suggests RDMA as an alternative.

RDMA: Remote Direct Memory Access

- **Reliable**
  Requests sent through “queue pairs”.
  Network failures terminate connections.

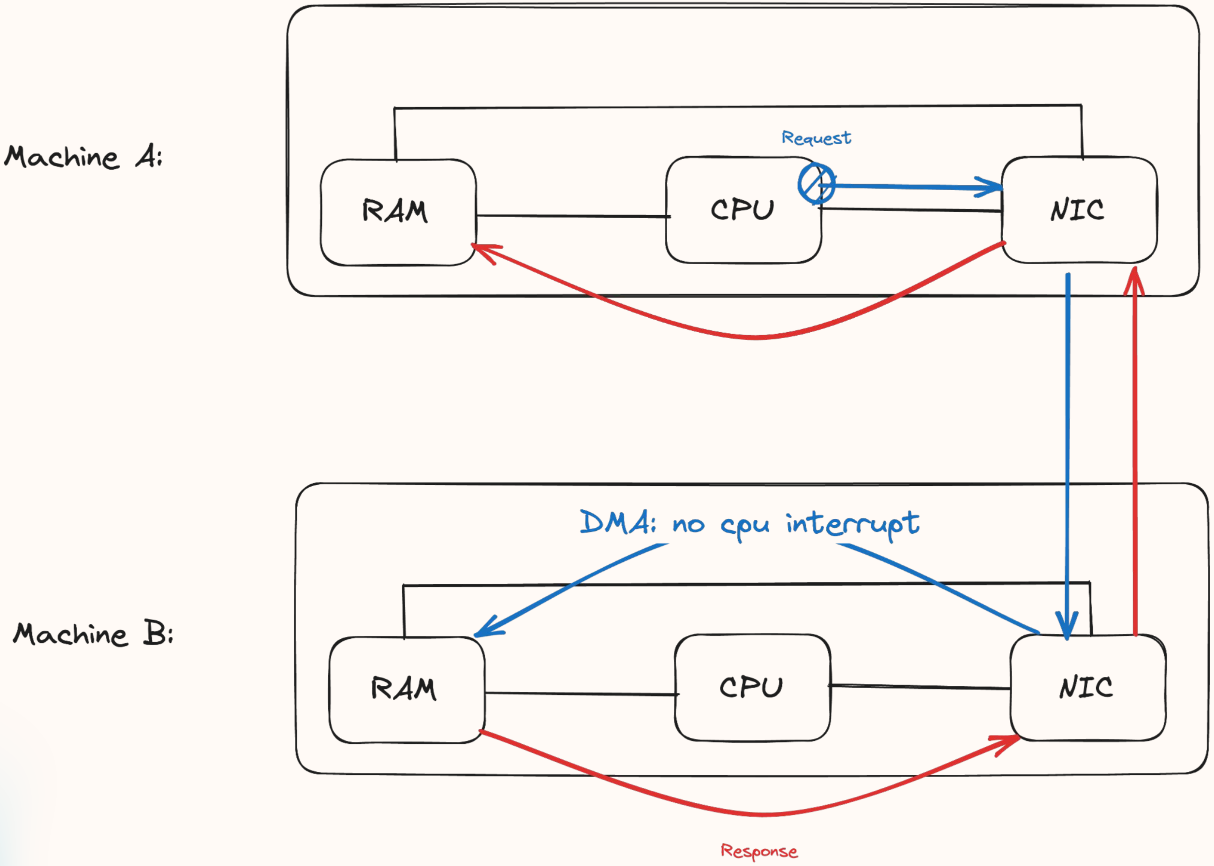
- **Fast**
  NICs perform requests.
  Bypasses kernel and remote cpu for read/write operations.
  No network stack overhead.

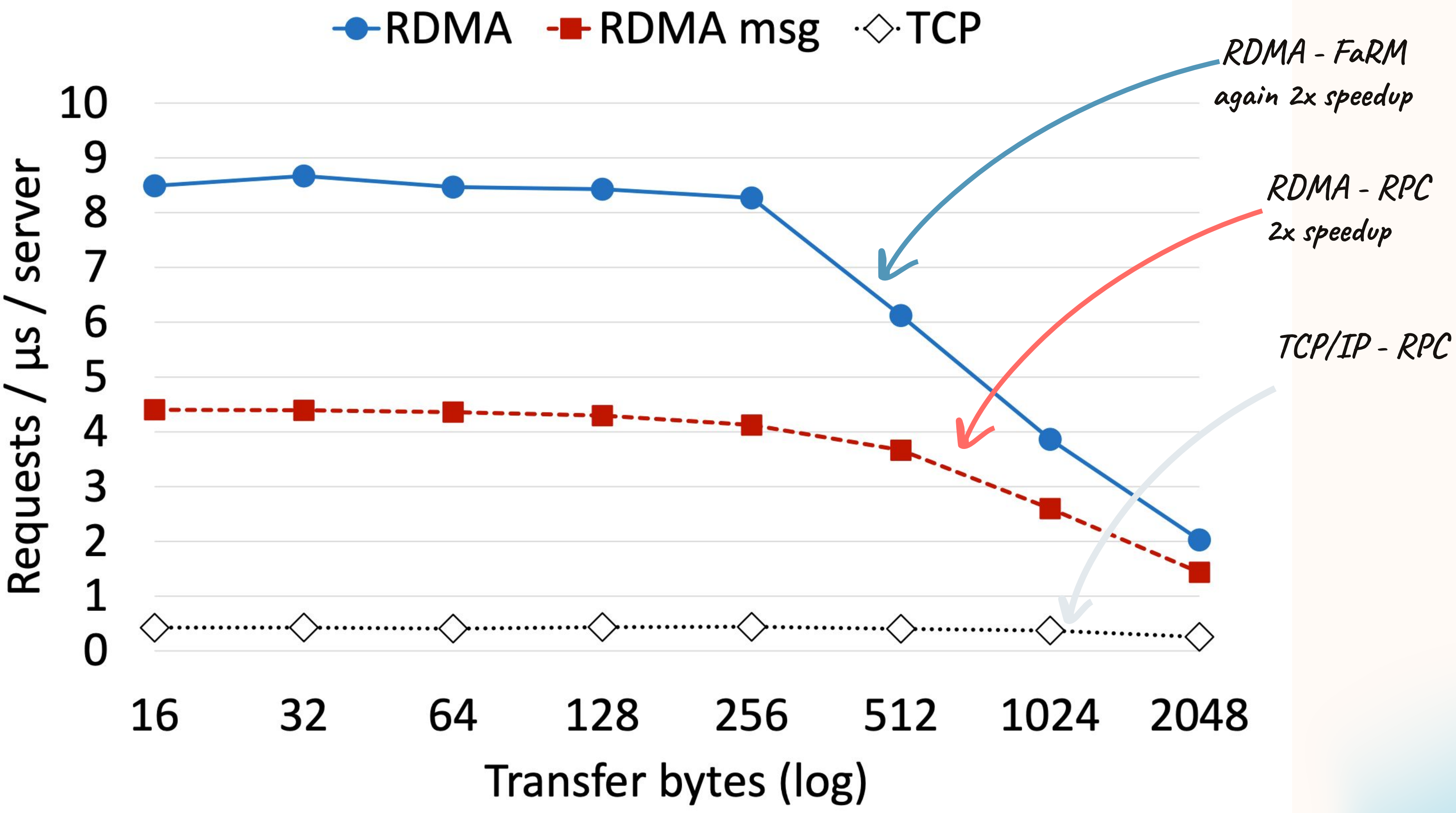
  100Gb/s throughput
  1–3 μs latency

- **Recently became affordable through RoCE**
  = RDMA over Converged Ethernet

  Allows RDMA over Ethernet instead of expensive “Infiniband” cables.
  Widely available data center bridging extensions.

RDMA message passing example:




RDMA
RDMA msg
TCP
Requests / µs / server
10
9
8
7
6
5
4
3
2
1
0
16
32
64
128
256
512
1024
2048
Transfer bytes (log)
RDMA - FaRM
again 2x speedup
RDMA - RPC
2x speedup
TCP/IP - RPC

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

Summary by Yahya Jabary for CS 854

FaRM: Fast Remote Memory

“FaRM is [...] a [main memory] distributed computing platform for modern data center hardware”

– **Aleksandar Dragojević**

"FaRM is [...] a [main memory] **distributed computing platform** for modern data center hardware"

– **Aleksandar Dragojević**

*Basically a **library** to program distributed systems in the future in.*

Use-cases

Distributed high performance / low latency applications:

- **Deep neural networks**
- Scale-out OLTP (online transaction processing)
- **Graph processing**
- **Distributed databases**

  Authors built a key-value / graph store similar to Facebook's "Tao":

  - 167 million key-value lookups/second
  - Average 31μs latency

FaRM API

1. **Shared memory abstractions**
   Accessing remote memory as if it would be local.
2. **Memory synchronization**
   Enforcing data consistency.
3. **Communication**
   RDMA based message passing interface (MPI)

```
// transaction
Tx*  txCreate(); // starts transaction, returns context
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);

// lock free distributed transactions
Lf*  lockFreeStart();
void lockFreeRead(Lf* op, Addr a, int size,Cont *c);
void lockFreeEnd(Lf *op);

Incarnation objGetIncarnation(ObjBuf *o);
void objIncrementIncarnation(ObjBuf *o);

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

FaRM API

1. **Shared memory abstractions**
   Accessing remote memory as if it would be local.

2. **Memory synchronization**
   Enforcing data consistency.

3. **Communication**
   RDMA based message passing interface (MPI)

```
// transaction
Tx*  txCreate(); // starts transaction, returns context
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);

// lock free distributed transactions
Lf*  lockFreeStart();
void lockFreeRead(Lf* op, Addr a, int size,Cont *c);
void lockFreeEnd(Lf *op);

Incarnation objGetIncarnation(ObjBuf *o);
void objIncrementIncarnation(ObjBuf *o);

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

Shared memory abstractions

Shared address space

= partitioned global address space (PGAS)

The entire cluster’s memory is shared.

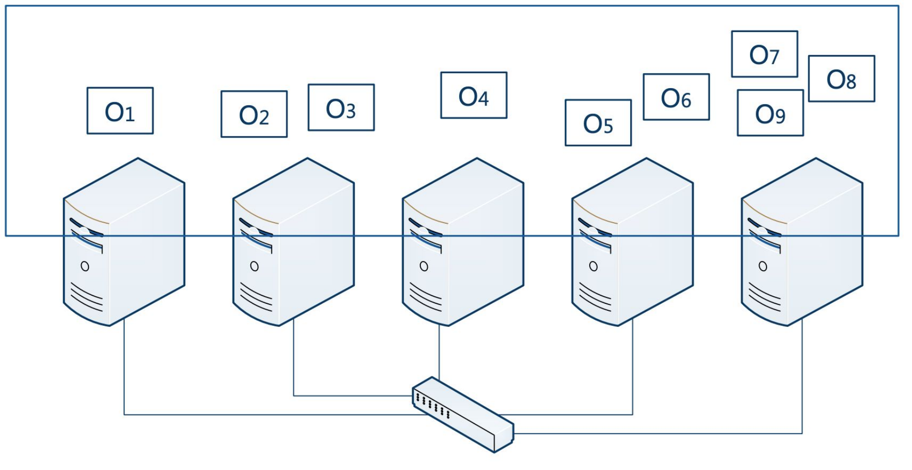
Necessary for efficient RDMA.

Convenient for programming.

Shared memory abstractions

Shared address space

= partitioned global address space (PGAS)



The entire cluster's memory is shared.

Necessary for efficient RDMA.

Convenient for programming.

Shared memory abstractions

Distributed transactions

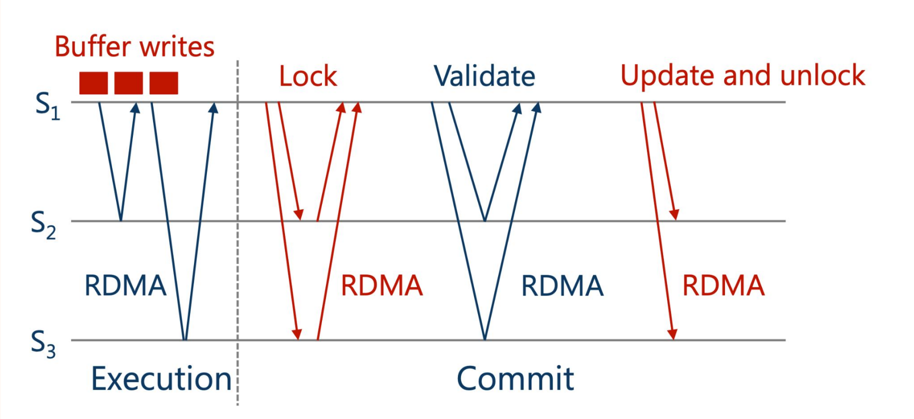
= atomic execution of multiple operations over the shared address

Operations: read, write, allocate, free

- ACID transactions
- Fault tolerance
- Strict serializability
- 2-Phase commit protocol
  with RDMA-based messaging

General primitive of FaRM.

Shared memory abstractions



2 phase commit

Distributed transactions

= atomic execution of multiple operations over the shared address

Operations: read, write, allocate, free

- ACID transactions
- Fault tolerance
- Strict serializability
- 2-Phase commit protocol
  with RDMA-based messaging

General primitive of FaRM.

Lock-free reads

Traditional writes

Memory on a single machine.

Write:

1. Lock counter
2. Update data
3. Unlock and increment counter

Version:
64 bit counter in header to detect inconsistencies

Lock-free reads

Traditional lock free reads

Memory on a single machine.

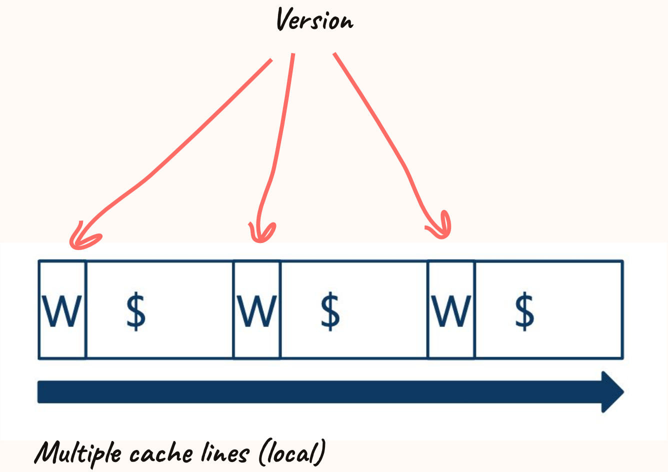
Read:

1. Read counter
2. Read data
3. Read counter again, check if it matches

3 RDMA operations, too expensive

Version:
64 bit counter in header to detect inconsistencies

Lock-free reads



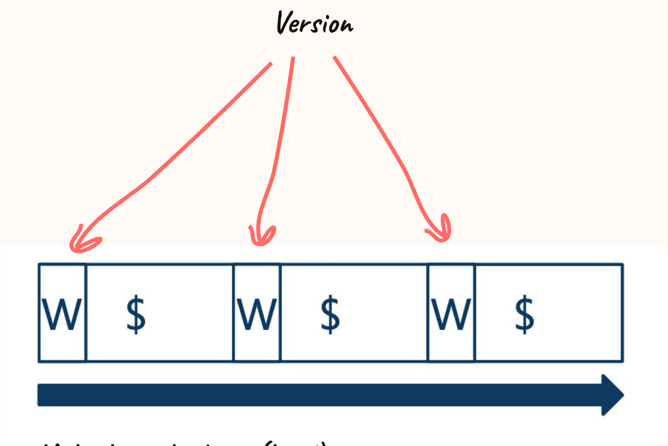
FaRM lock-free reads

Distributed memory.
16 bit version for space efficiency.

Write:

1. Lock all versions
2. Update object
   (spread across multiple cache lines)
3. Unlock and increment

Lock-free reads



Multiple cache lines (local)

FaRM lock-free reads

Distributed memory.
16 bit version for space efficiency.

Read: Just measure operation time

$\Delta_{write\_minimum} = 40ns$
$\Delta_{read\_maximum} = 40ns \cdot 2^{16} \cdot (1 - \varepsilon) = 2ms$

delay tolerance

Check if versions match locally on single read RDMA.

Performance Optimizations

A final overview of the most essential optimizations in FaRM.

Major performance improvements

1) Locality Awareness

“Function shipping”:

Sending instructions to be executed as close to data as possible.

2) Custom NIC drivers

NIC was running out of cache space.

PhyCo = kernel driver that allocates 2GB contiguous and aligned memory regions.

3) Connection multiplexing

Sharing remote messaging ‘queue pairs’ among multiple threads on remote machines.

Lower memory usage.
But also lower parallelism.

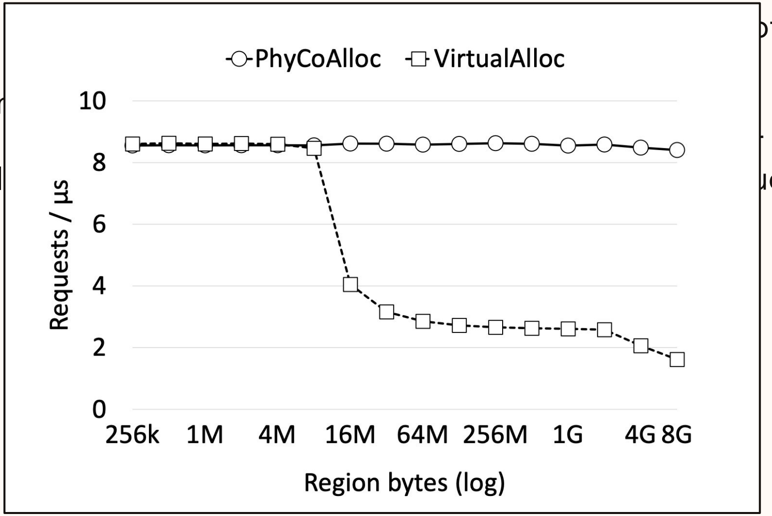
Will be solved in future papers.

Major performance improvements

1) Locality Awareness

“Function

Sending i
executed
as possibl



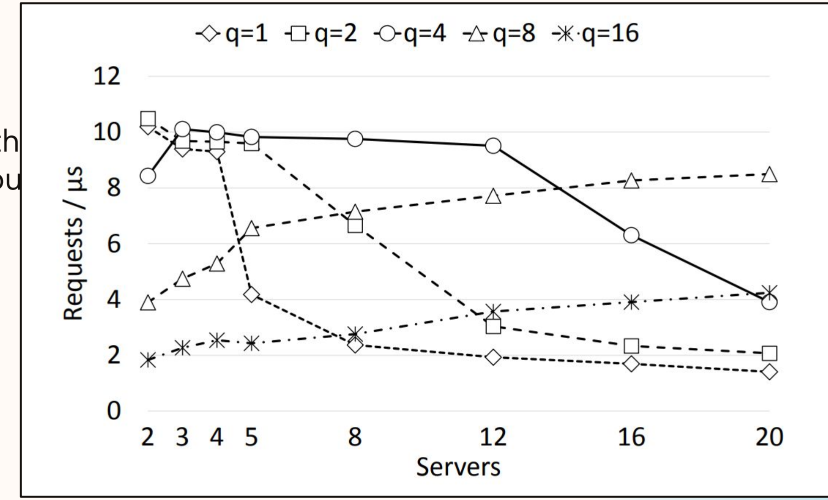
2) Custom NIC drivers

of

th

uou

3) Connection multiplexing



Thanks!

Do you have **any questions**?