FaRM

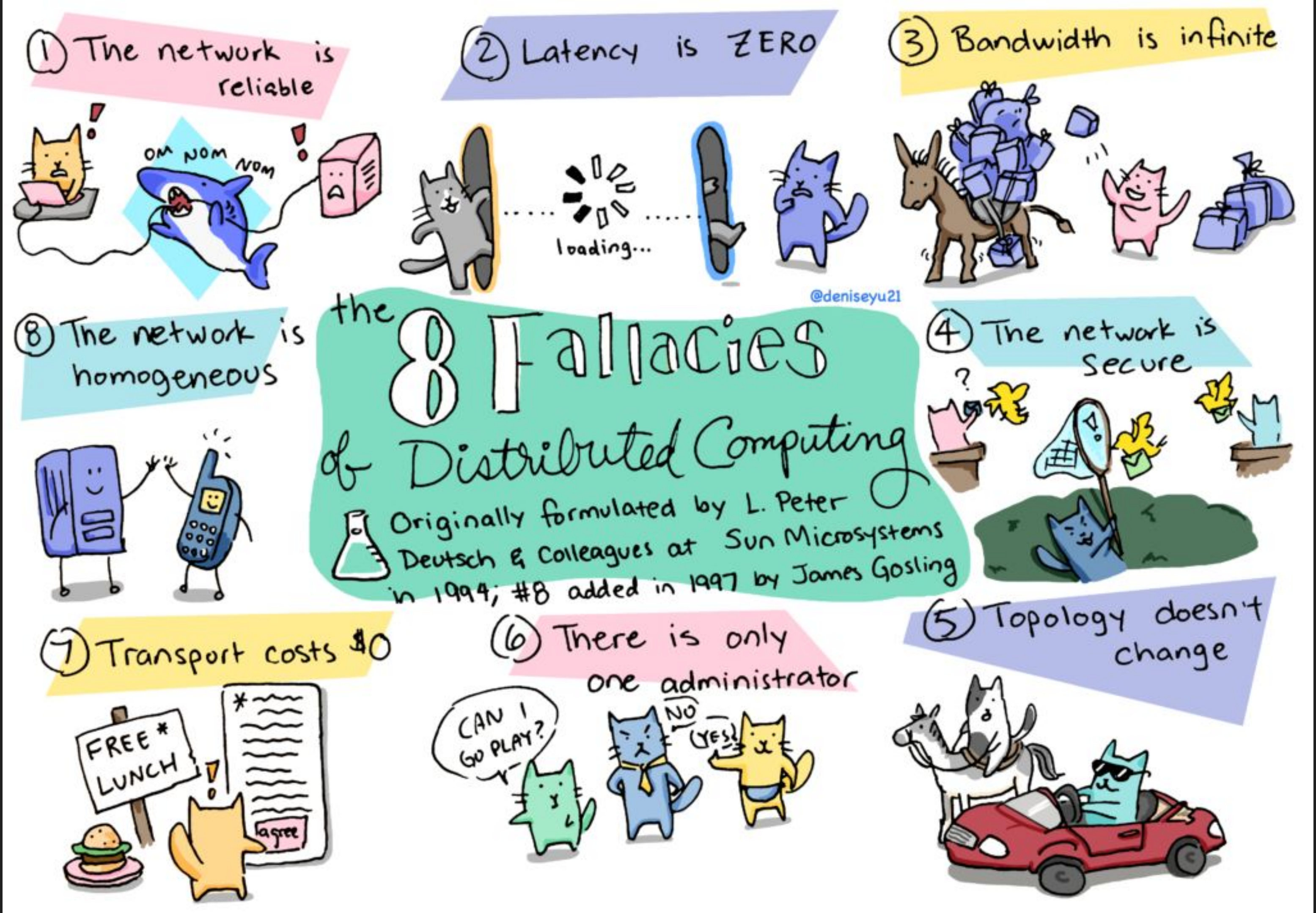
Presented by Jiaxiao Zhou (周佳孝 Mossaka)

Microsoft

Policy Against Harassment at ACM Activities

https://www.acm.org/about-acm/policy-against-harassment

MSFT Sys Meetup wants to encourage and preserve this open exchange of ideas, which requires an environment that enables all to participate without fear of personal harassment. We define harassment to include specific unacceptable factors and behaviors listed in the ACM’s policy against harassment. Unacceptable behavior will not be tolerated.


① The network is reliable
OM NOM NOM
② Latency is ZERO
loading...
③ Bandwidth is infinite
@deniseyu21
the 8 Fallacies of Distributed Computing
Originally formulated by L. Peter Deutsch & Colleagues at Sun Microsystems in 1994; #8 added in 1997 by James Gosling
⑧ The network is homogeneous
④ The network is secure
?
⑦ Transport costs $0
FREE* LUNCH
*
I agree
⑥ There is only one administrator
CAN I GO PLAY?
NO
YES!
⑤ Topology doesn't change

[(7) Denise Yu (@deniseyu21) / Twitter]

Agenda

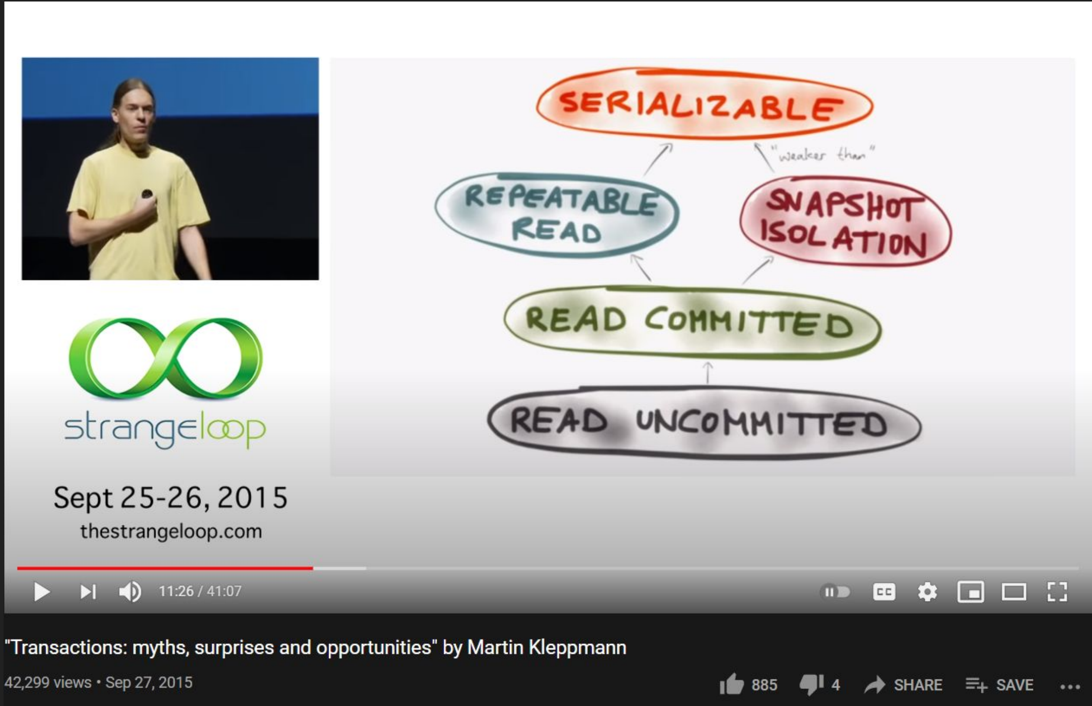
1. Distributed Transaction Overview
2. What is FaRM
3. Hardware Trends
4. FaRM architecture
5. OCC and FaRM commit protocol
6. Failure Recovery
    a. Failure detection
    b. Reconfiguration
    c. Transaction state recovery
7. Evaluation and Summary

Distributed Transactions

Distributed transaction = concurrency control + atomic commit

Correct behavior: *ACID*

Isolation levels:



"Transactions: myths, surprises and opportunities" by Martin Kleppmann

42,299 views • Sep 27, 2015

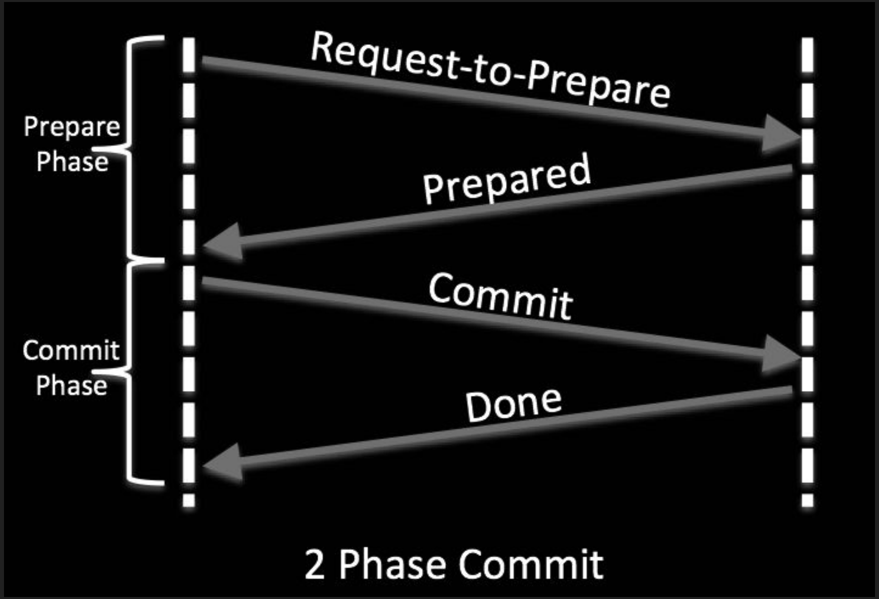
Distributed Transactions

Two classes of concurrency control for Tx

1. Pessimistic (2PL)
2. Optimistic (today)

Atomic Commit:

Two-phase commit



Last time

Google Spanner:

1. 2PC over Paxos
2. external consistency
3. geographic replication
4. TrueTime

Microsoft

No compromises: distributed transactions with consistency, availability, and performance

Aleksandar Dragojević, Dushyanth Narayanan, Edmund B. Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, Miguel Castro

Microsoft Research

Aleksandar Dragojevic

Senior Researcher

Abstract

Transactions with strong consistency and high availability simplify building and reasoning about distributed systems. However, previous implementations performed poorly. This forced system designers to avoid transactions completely, to weaken consistency guarantees, or to provide single-machine transactions that require programmers to partition their data. In this paper, we show that there is no need to compromise in modern data centers. We show that a main memory distributed computing platform called FaRM can provide distributed transactions with strict serializability, high performance, durability, and high availability. FaRM achieves a peak throughput of 140 million TATP transactions per second on 90 machines with a 4.9 TB database, and it recovers from a failure in less than 50 ms. Key to achieving these results was the design of new transaction, replication, and recovery protocols from first principles to leverage commodity networks with RDMA and a new, inexpensive approach to providing non-volatile DRAM.

1. Introduction

Transactions with high availability and strict serializabil-

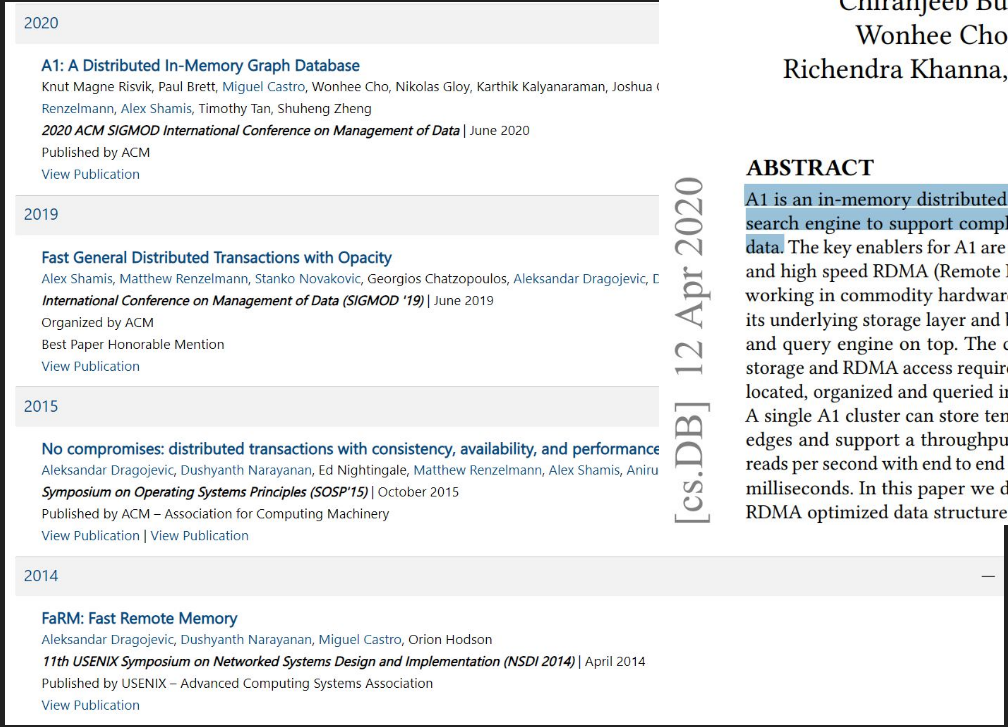
6, 9, 28]), provide transactions only when all the data resides within a single machine, forcing programmers to partition their data and complicating reasoning about correctness.

This paper demonstrates that new software in modern data centers can eliminate the need to compromise. It describes the transaction, replication, and recovery protocols in FaRM [16], a main memory distributed computing platform. FaRM provides distributed ACID transactions with strict serializability, high availability, high throughput and low latency. These protocols were designed from first principles to leverage two hardware trends appearing in data centers: fast commodity networks with RDMA and an inexpensive approach to providing non-volatile DRAM. Non-volatility is achieved by attaching batteries to power supply units and writing the contents of DRAM to SSD when the power fails. These trends eliminate storage and network bottlenecks, but they also expose CPU bottlenecks that limit their performance benefit. FaRM's protocols follow three principles to address these CPU bottlenecks: *reducing message counts*, *using one-sided RDMA reads and writes instead of messages*, and *exploiting parallelism effectively*.

FaRM scales out by distributing objects across the machines in a data center while allowing transactions to span

The FaRM project is an active research project

FaRM - Microsoft Research



2020

A1: A Distributed In-Memory Graph Database
Knut Magne Risvik, Paul Brett, Miguel Castro, Wonhee Cho, Nikolas Gloy, Karthik Kalyanaraman, Joshua
Renzelmann, Alex Shamis, Timothy Tan, Shuheng Zheng
2020 ACM SIGMOD International Conference on Management of Data | June 2020
Published by ACM
View Publication

2019

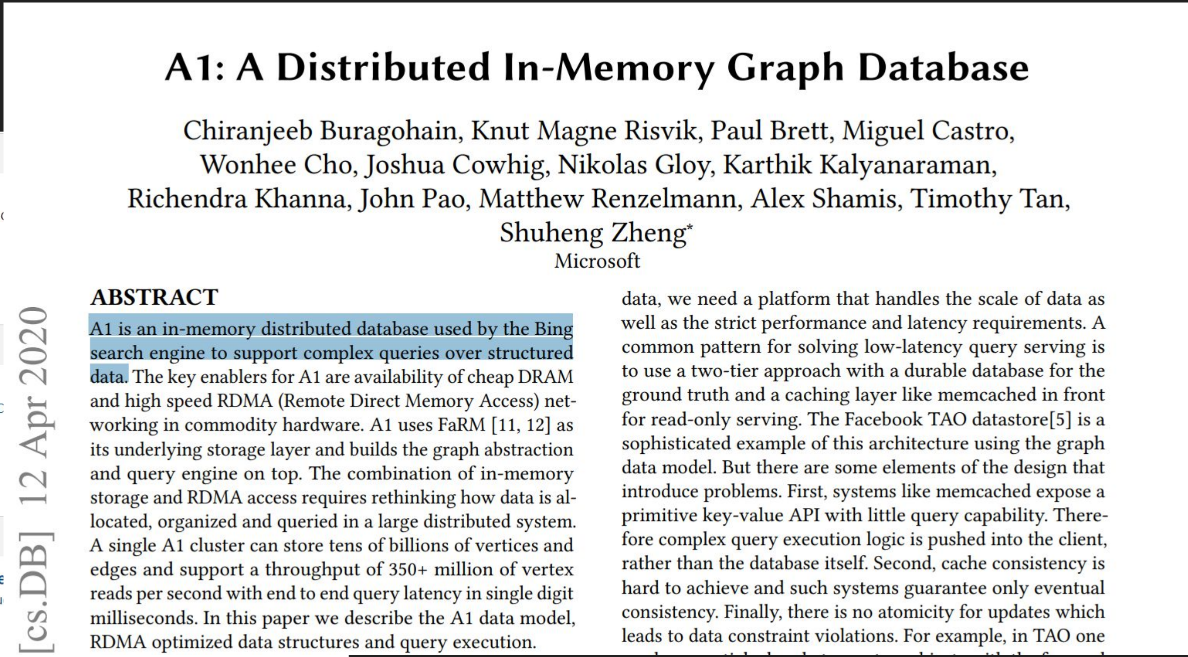
Fast General Distributed Transactions with Opacity
Alex Shamis, Matthew Renzelmann, Stanko Novakovic, Georgios Chatzopoulos, Aleksandar Dragojevic,
International Conference on Management of Data (SIGMOD '19) | June 2019
Organized by ACM
Best Paper Honorable Mention
View Publication

2015

No compromises: distributed transactions with consistency, availability, and performance
Aleksandar Dragojevic, Dushyanth Narayanan, Ed Nightingale, Matthew Renzelmann, Alex Shamis, Aniru
Symposium on Operating Systems Principles (SOSP'15) | October 2015
Published by ACM – Association for Computing Machinery
View Publication | View Publication

2014 —

FaRM: Fast Remote Memory
Aleksandar Dragojevic, Dushyanth Narayanan, Miguel Castro, Orion Hodson
11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2014) | April 2014
Published by USENIX – Advanced Computing Systems Association
View Publication



A1: A Distributed In-Memory Graph Database

Chiranjeeb Buragohain, Knut Magne Risvik, Paul Brett, Miguel Castro, Wonhee Cho, Joshua Cowhig, Nikolas Gloy, Karthik Kalyanaraman, Richendra Khanna, John Pao, Matthew Renzelmann, Alex Shamis, Timothy Tan, Shuheng Zheng*
Microsoft

[cs.DB] 12 Apr 2020

ABSTRACT

A1 is an in-memory distributed database used by the Bing search engine to support complex queries over structured data. The key enablers for A1 are availability of cheap DRAM and high speed RDMA (Remote Direct Memory Access) networking in commodity hardware. A1 uses FaRM [11, 12] as its underlying storage layer and builds the graph abstraction and query engine on top. The combination of in-memory storage and RDMA access requires rethinking how data is allocated, organized and queried in a large distributed system. A single A1 cluster can store tens of billions of vertices and edges and support a throughput of 350+ million of vertex reads per second with end to end query latency in single digit milliseconds. In this paper we describe the A1 data model, RDMA optimized data structures and query execution.

data, we need a platform that handles the scale of data as well as the strict performance and latency requirements. A common pattern for solving low-latency query serving is to use a two-tier approach with a durable database for the ground truth and a caching layer like memcached in front for read-only serving. The Facebook TAO datastore[5] is a sophisticated example of this architecture using the graph data model. But there are some elements of the design that introduce problems. First, systems like memcached expose a primitive key-value API with little query capability. Therefore complex query execution logic is pushed into the client, rather than the database itself. Second, cache consistency is hard to achieve and such systems guarantee only eventual consistency. Finally, there is no atomicity for updates which leads to data constraint violations. For example, in TAO one

What is FaRM?

FaRM is a transactional distributed in-memory storage system.

1. Provides *ACID* transactions with strict serializability, high availability, and high throughput and low latency.
2. FaRM is faaaaaaast. Performance is 50 us (100x faster than Spanner) and 140 million TATP transactions per second on 90 machines
3. Explores the new hardware trends

Pause for a moment… Let’s imagine we want to design a high-performance system, what are the bottlenecks? 🤔

FaRM solves

Pause for a moment… Let's imagine we want to design a high-performance system, what are the bottlenecks? 🤔

1. Networks ✔️
2. Disk R/W ✔️
3. CPU ❌

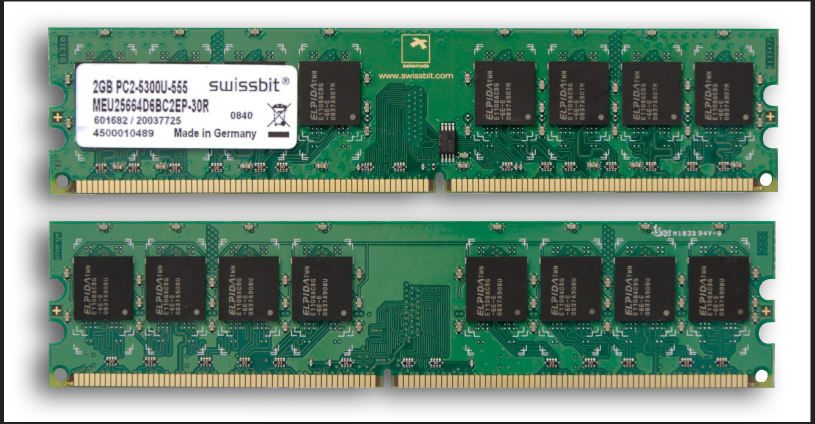
Hardware Trends

1. Lots of cheap DRAM available (256 GB RAM per machine)
2. Non-volatile memory, backed up by li-ion batteries
3. Fast networks with RDMA (remote direct memory access)
4. Kernel bypassing

Hardware Trends

1. Lots of cheap DRAM available (256 GB RAM per machine)
2. **Non-volatile memory, backed up by li-ion batteries**
3. **Fast networks with RDMA (remote direct memory access)**
4. **Kernel bypassing**

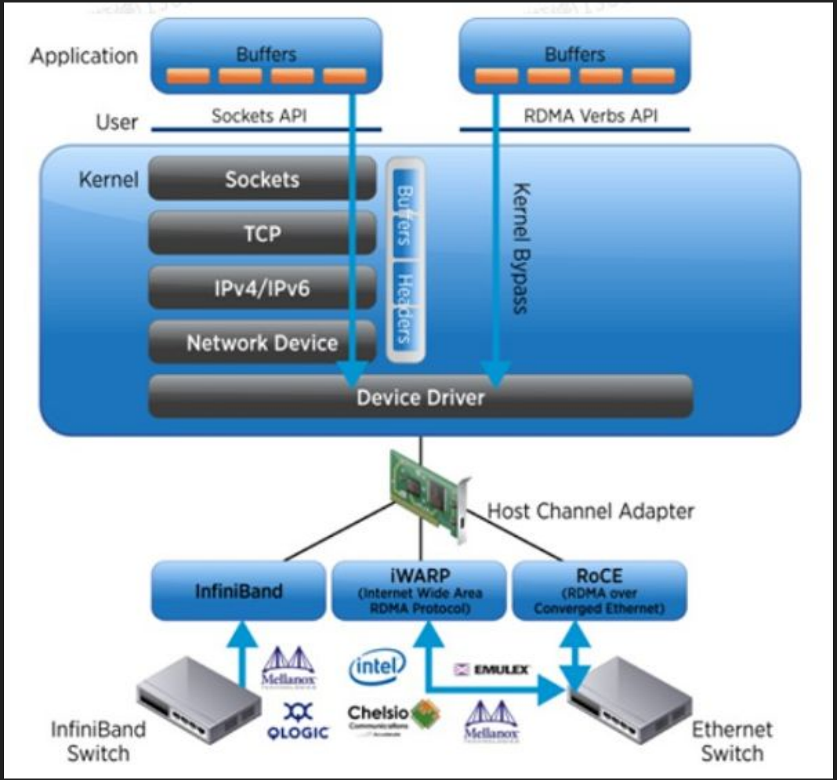
NVRAM (non-volatile RAM)

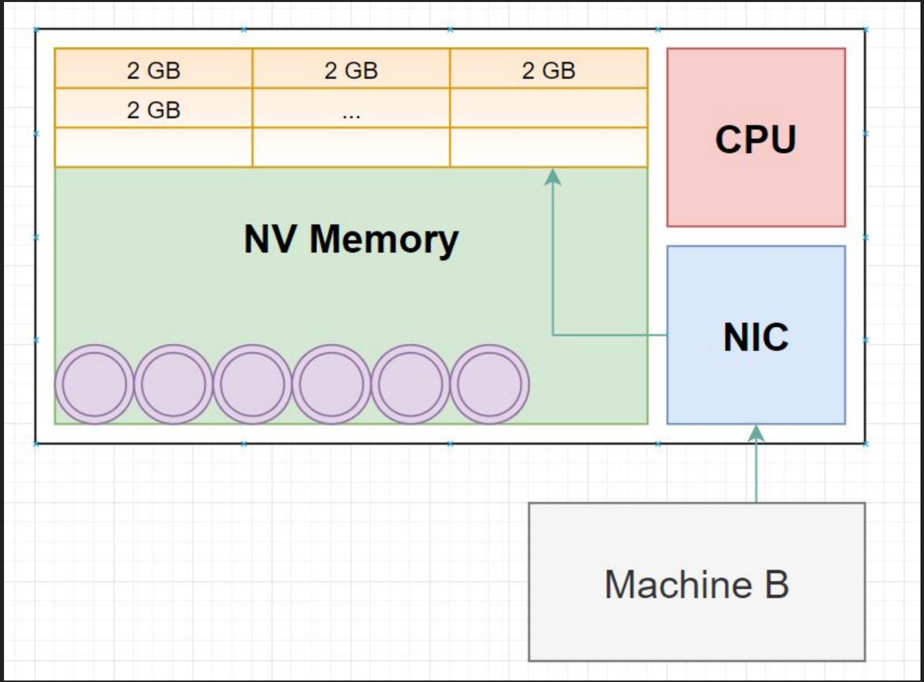


Why in memory? RAM write takes 200 ns, SSK write take 10 us.

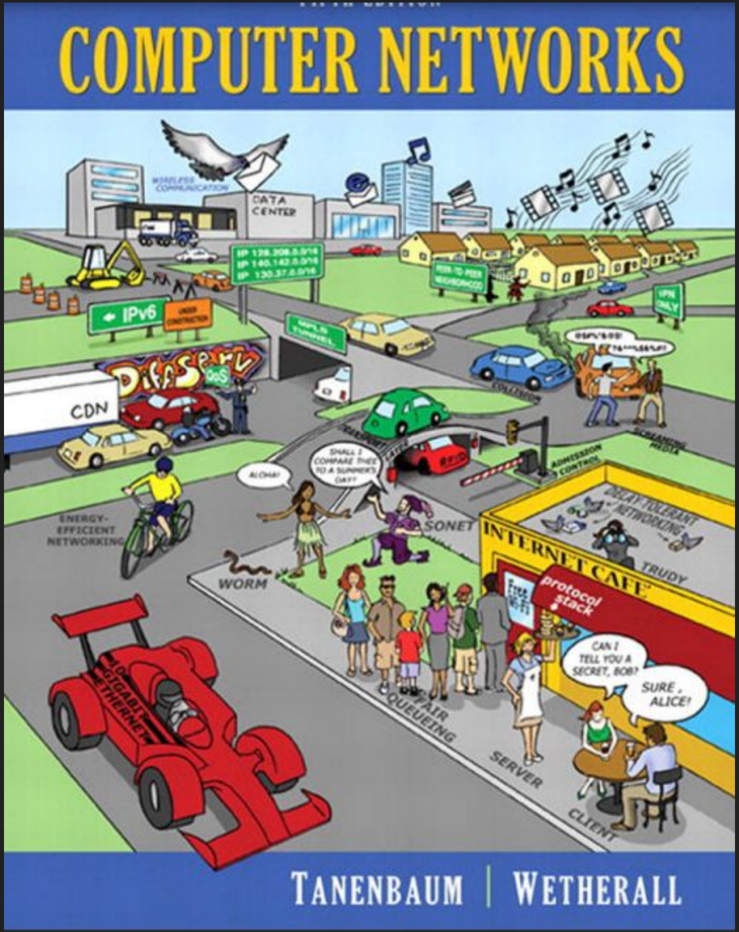
Eliminates the persistence write bottleneck

RDMA and Kernel Bypassing





RDMA and Kernel Bypassing



RDMA and Kernel Bypassing

Eliminates the network bottleneck

100 Gbps of bandwidth

100 Million ops/s, 1-3 us latency

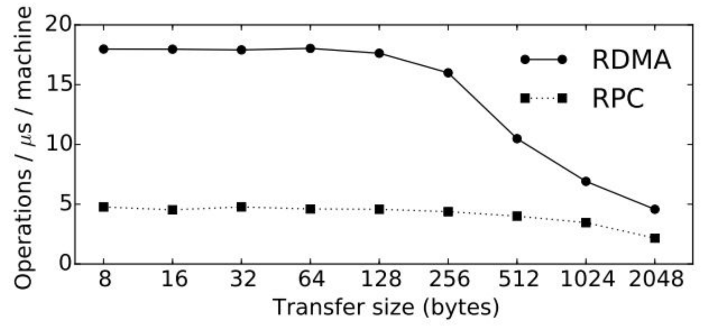
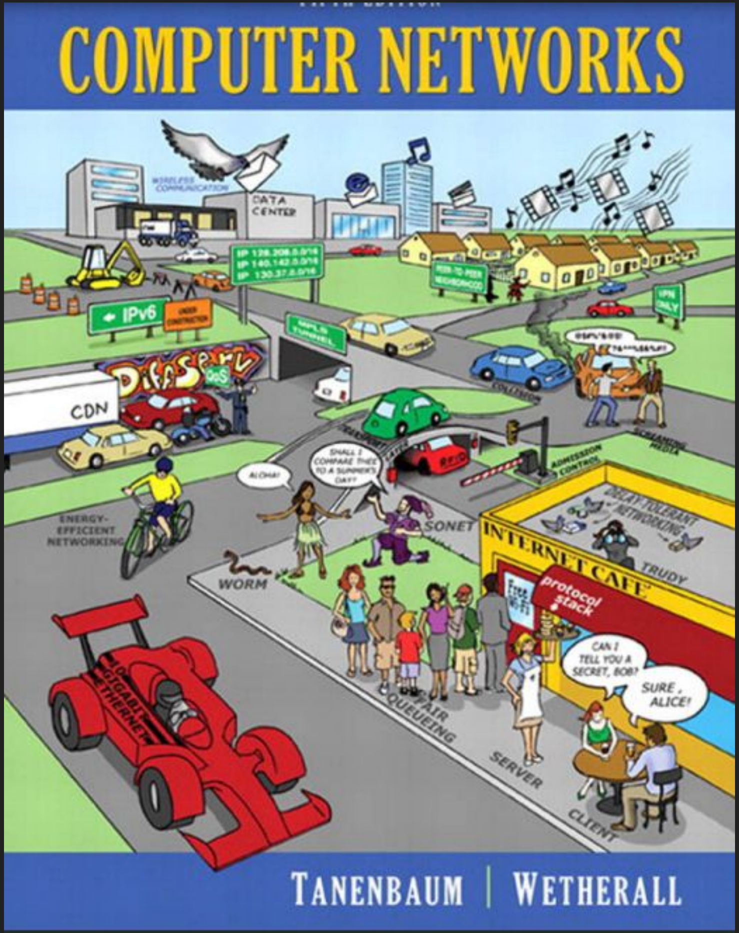


Figure 2. Per-machine RDMA and RPC read performance



The Challenge

RDMA is one-sided, meaning remote CPU is not involved. How are we going to redesign distributed transactional protocols? 🤔

It seems that transaction and RDMA are not compatible

1. Locking
2. Validation
3. Committing

FaRM Architecture

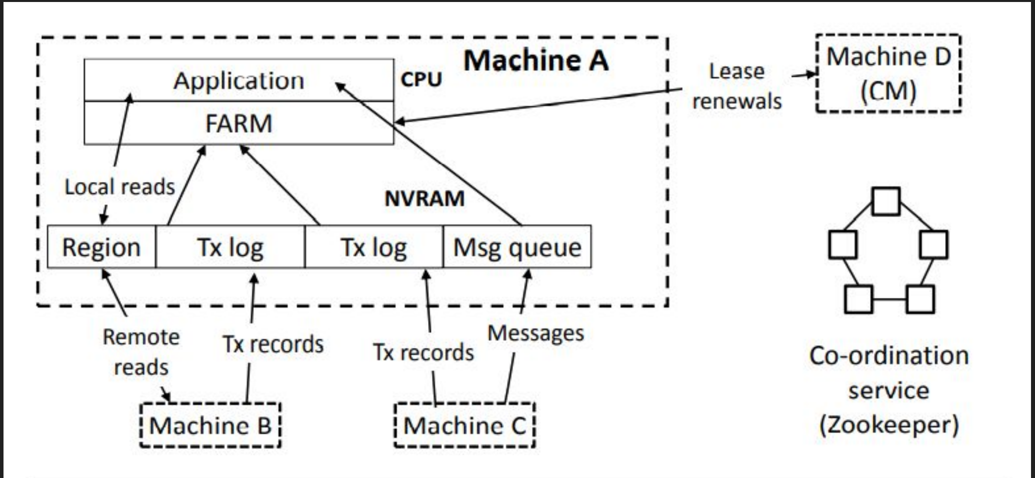


Figure 3. FaRM architecture

Shared memory

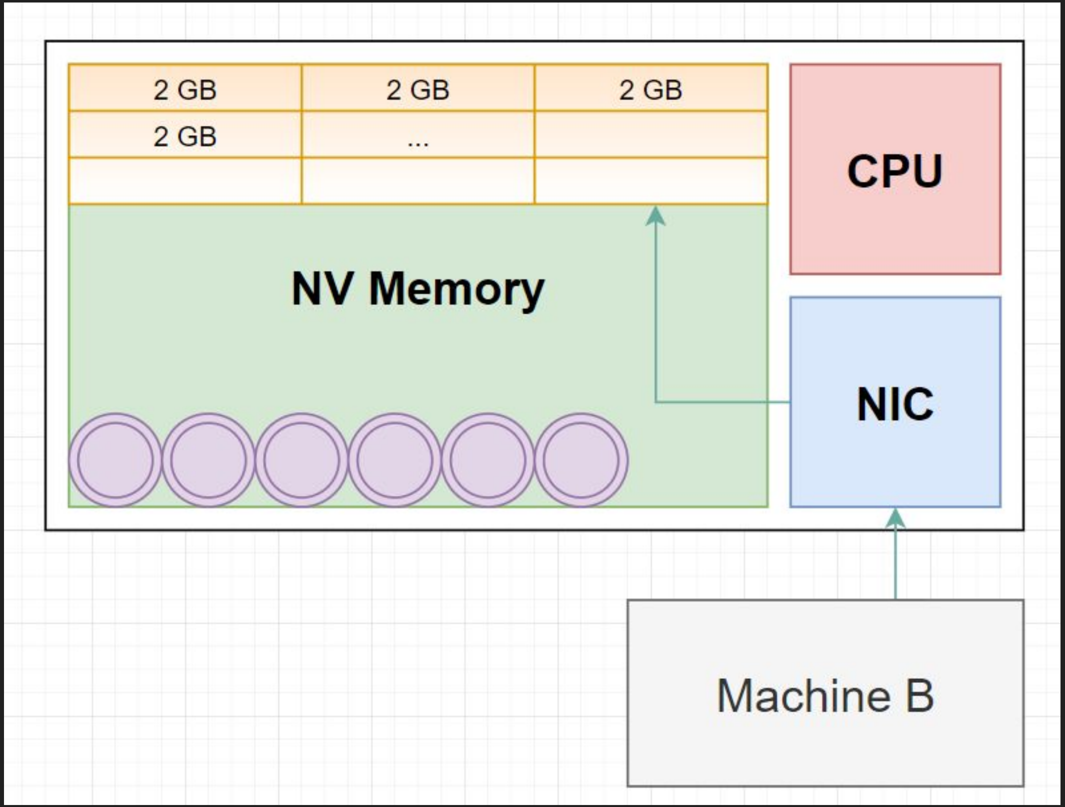
Transactions read, write, alloc, free

Symmetric model

Machine executes client and server code

Primary/Backup

FaRM Architecture



OCC

Updates are buffered locally during execution and only made visible to other transactions on a successful commit.

Serializable

Lock-free reads

Q: What is CM?

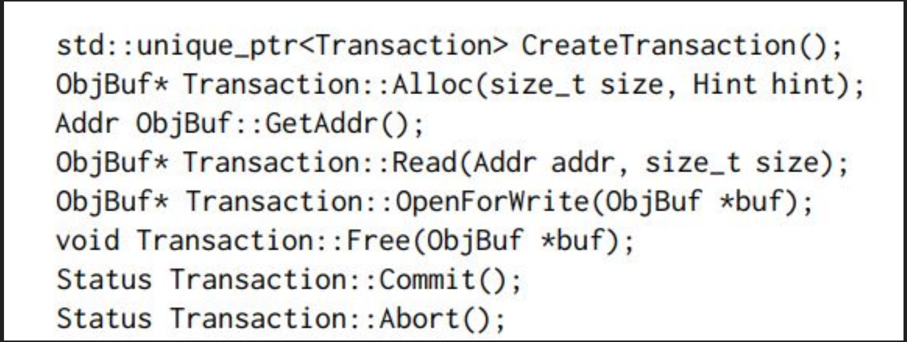
A: configuration manager that manages leases, detect failures, and coordinate recovery. It also stores mapping from region id to its primary, backups

Q: **Why Zookeeper is needed?**

A: Coordination service to ensure machines agree on config

Q: **What does FaRM API look like?**

A: No SQL 😥



```
std::unique_ptr<Transaction> CreateTransaction();
ObjBuf* Transaction::Alloc(size_t size, Hint hint);
Addr ObjBuf::GetAddr();
ObjBuf* Transaction::Read(Addr addr, size_t size);
ObjBuf* Transaction::OpenForWrite(ObjBuf *buf);
void Transaction::Free(ObjBuf *buf);
Status Transaction::Commit();
Status Transaction::Abort();
```

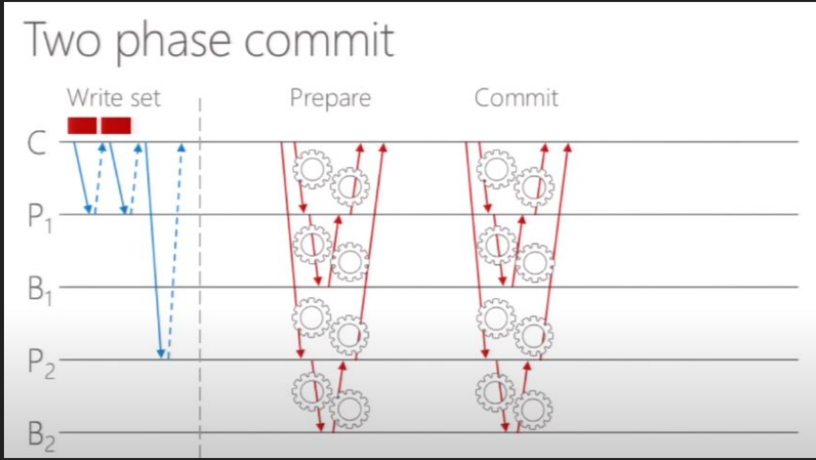
Distributed Transaction and Replication

DRAM is used both for data and Tx log

Primary/Backups and use TC in 4PC to communicate

Q: **Why not 2PC?**

A: FaRM talk mentions 2PC uses too many messages and require CPU overheads.



Distributed Transaction and Replication

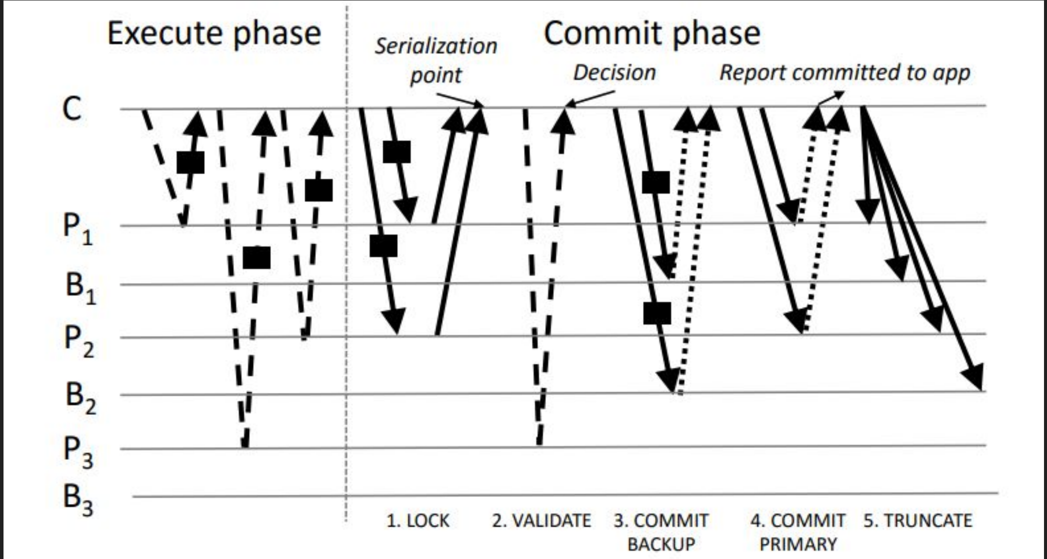
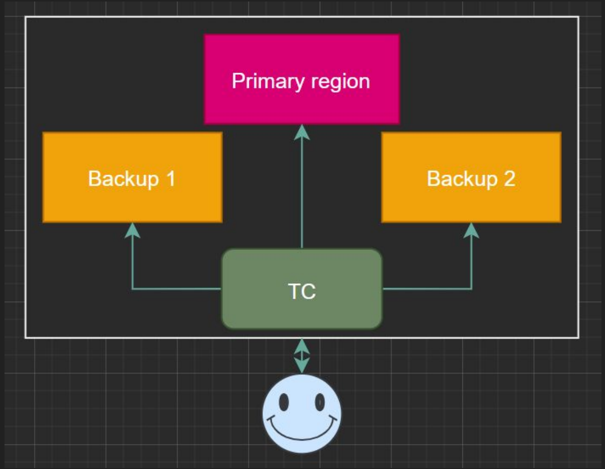


Figure 4. FaRM commit protocol with a coordinator C, primaries on P_1, P_2, P_3, and backups on B_1, B_2, B_3. P_1 and P_2 are read and written. P_3 is only read. We use dashed lines for RDMA reads, solid ones for RDMA writes, dotted ones for hardware acks, and rectangles for object data.



Q: Why is validate phase needed?

A: example

T1: `if x == 0; y = 1`

T2:`if y == 0; x = 1`

Serializable means: T1, T2 or T2, T1

If they run concurrently:

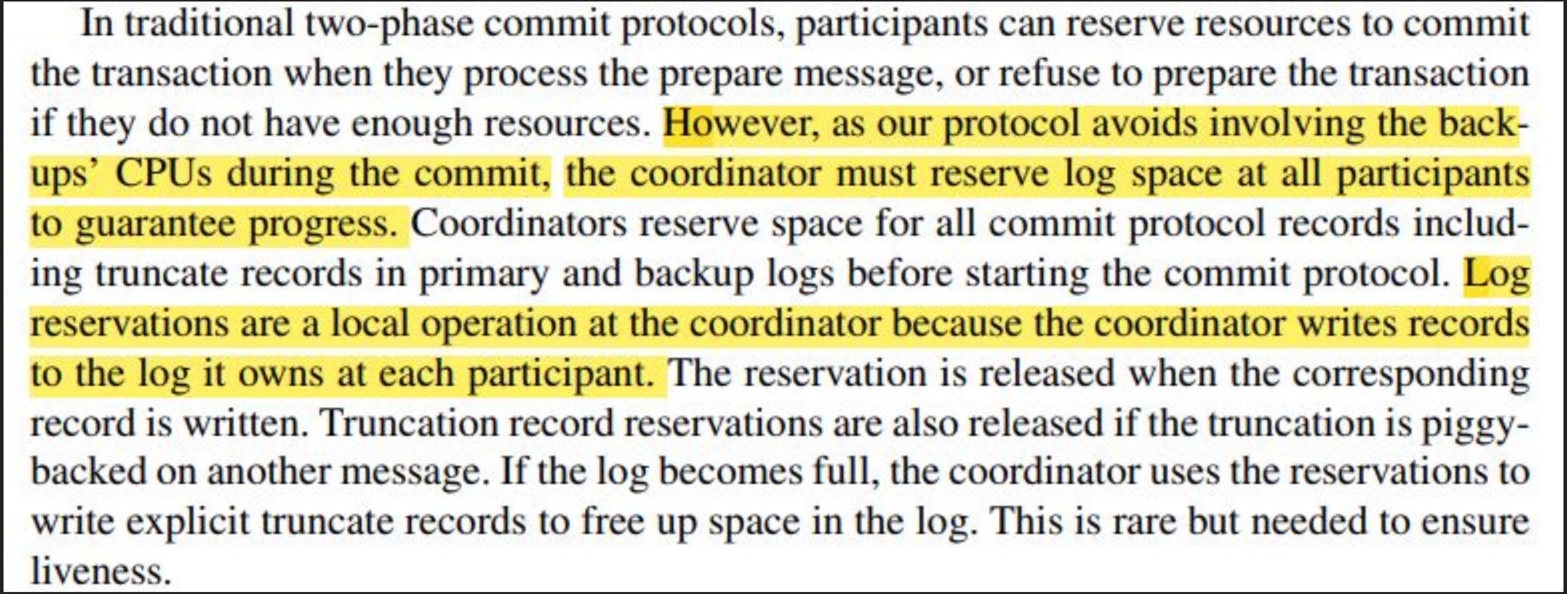
T1: Rx, Ly, Vx, Cy

T2: Ry, Lx, Vy, Cx

What will happen if validate phase is not there?

Q: Why is truncate phase needed?

A:



In traditional two-phase commit protocols, participants can reserve resources to commit the transaction when they process the prepare message, or refuse to prepare the transaction if they do not have enough resources. However, as our protocol avoids involving the backups' CPUs during the commit, the coordinator must reserve log space at all participants to guarantee progress. Coordinators reserve space for all commit protocol records including truncate records in primary and backup logs before starting the commit protocol. Log reservations are a local operation at the coordinator because the coordinator writes records to the log it owns at each participant. The reservation is released when the corresponding record is written. Truncation record reservations are also released if the truncation is piggybacked on another message. If the log becomes full, the coordinator uses the reservations to write explicit truncate records to free up space in the log. This is rare but needed to ensure liveness.

Q: How it’s performance compare to Spanner?

A: Spanner uses 2PC with Paxos to replicate TC. It requires 2f + 1 replicas to tolerate f failures and it requires 4P(2f+1) messages.

FaRM uses primary/backup replication, so it requires f + 1 copies to tolerate f failures. The commit phase uses Pw(f + 3) one-sided RMDA writes, and Pr RMDA reads.

A purely read-only FaRM transaction uses only RMDA reads, no writes, no log records - hence super faaaaaasst.

Correctness

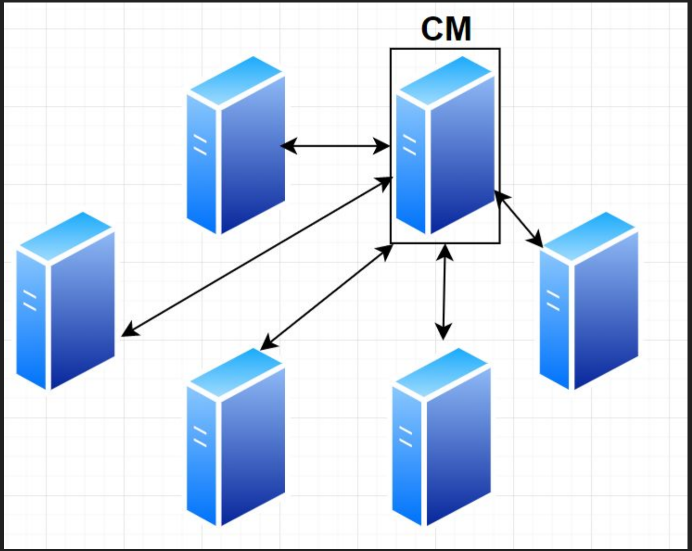
1. Committed R/W Txs are serializable at the point where all the writes locks are acquired.
2. Committed R/O Txs are serializable at the point of their last read.
3. Strict serializable: the serialization point is always between the start of execution and the completion.

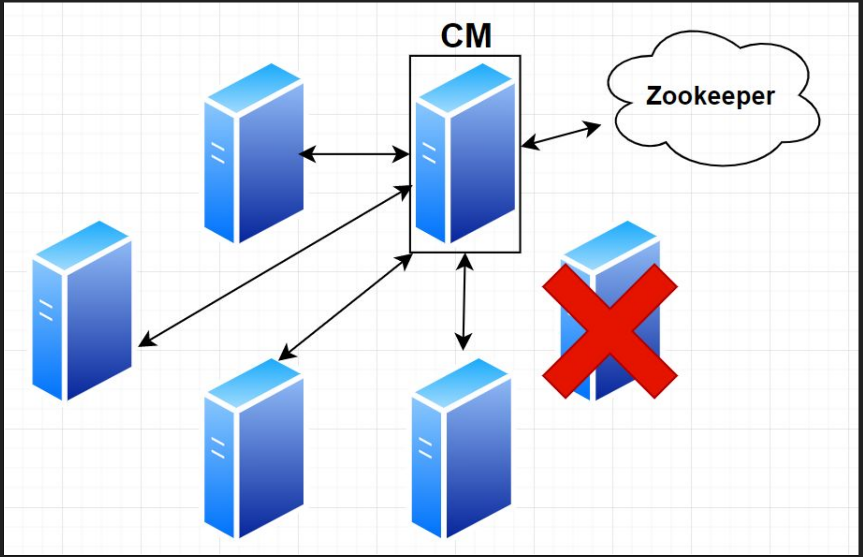
Agenda

1. Distributed Transaction Overview
2. What is FaRM
3. Hardware Trends
4. FaRM architecture
5. OCC and FaRM commit protocol
6. **Failure Recovery**
   a. Failure detection
   b. Reconfiguration
   c. Transaction state recovery

Failure detection

FaRM uses 5ms **leases** to detect failures.

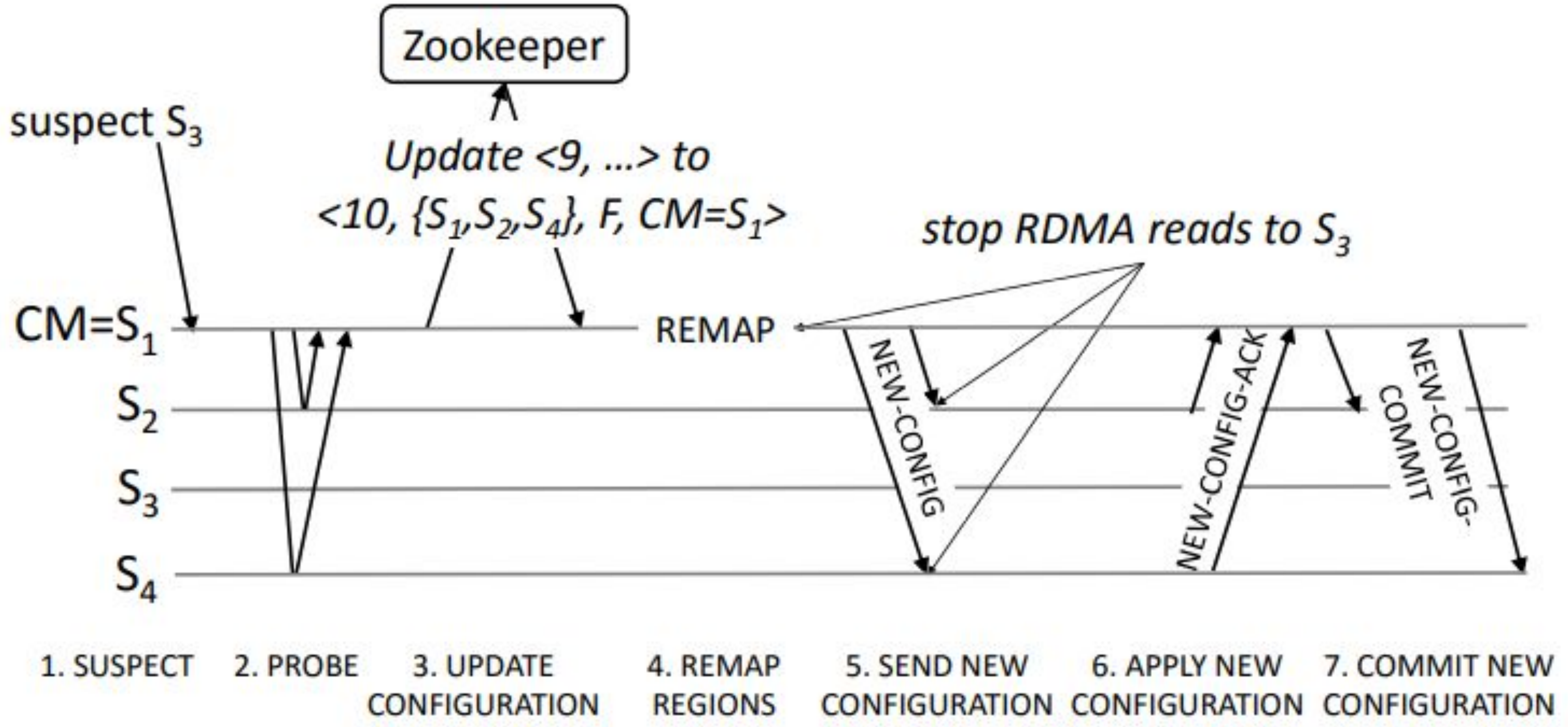




Reconfiguration Challenge

Q: RDMA reads have no server CPU involved. Then how does server know if it’s being evicted before replying to requests to access the object? 🤔

Precise Membership: after a failure, all machines in a new configuration must agree on its membership before allowing object mutations.



Zookeeper
suspect S_3
Update <9, …> to <10, {S_1,S_2,S_4}, F, CM=S_1>
stop RDMA reads to S_3
CM=S_1
S_2
S_3
S_4
REMAP
NEW-CONFIG
NEW-CONFIG-ACK
NEW-CONFIG-COMMIT
1. SUSPECT
2. PROBE
3. UPDATE CONFIGURATION
4. REMAP REGIONS
5. SEND NEW CONFIGURATION
6. APPLY NEW CONFIGURATION
7. COMMIT NEW CONFIGURATION

Figure 5. Reconfiguration

Q: What is REMAP and why is it needed?

A: The new CM reassigns regions previously mapped to failed machines to restore the replicas to f + 1. It always promotes a surviving backup to be the new primary.

Q: What does a config look like?

A: A config is of the form `<c, S, F, CM>`. When the new CM receives replies to the probes, it attempts to update config to `<c+1, S, F, CM_me>`

Transaction state recovery

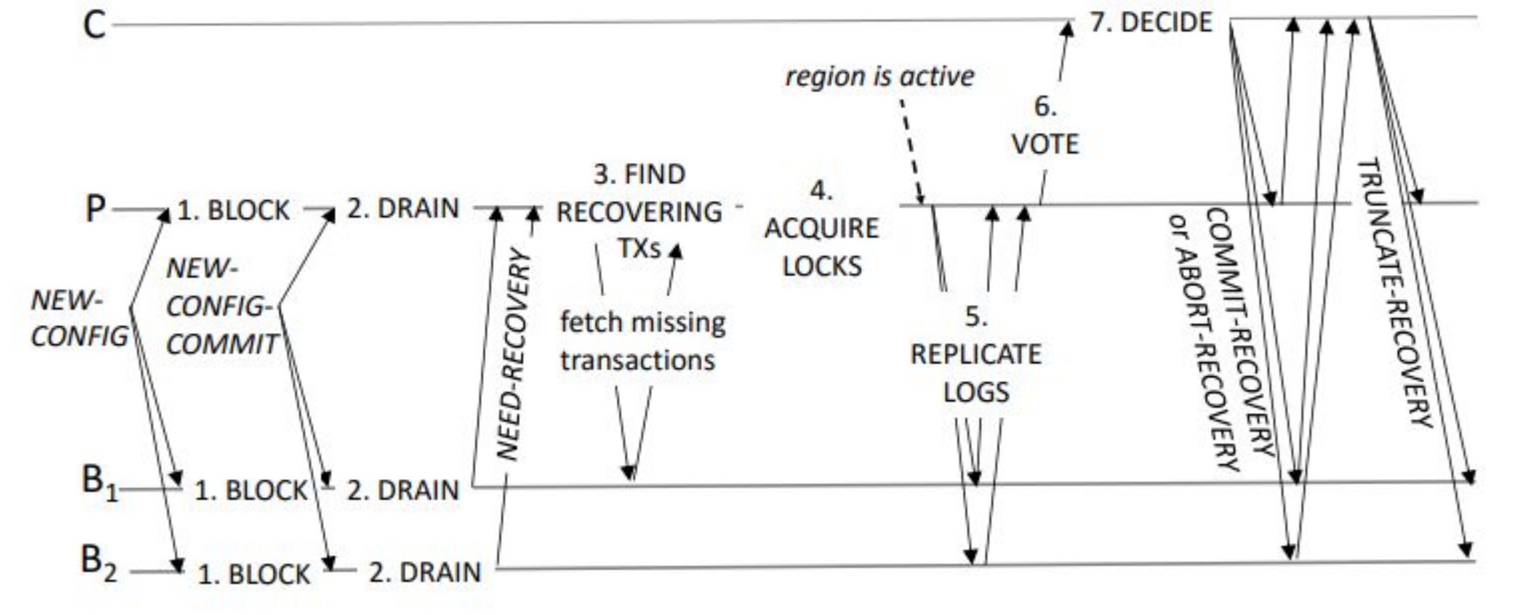


Figure 6. Transaction state recovery showing a coordinator C, primary P, and two backups B_1 and B_2

Q: Why draining is needed?

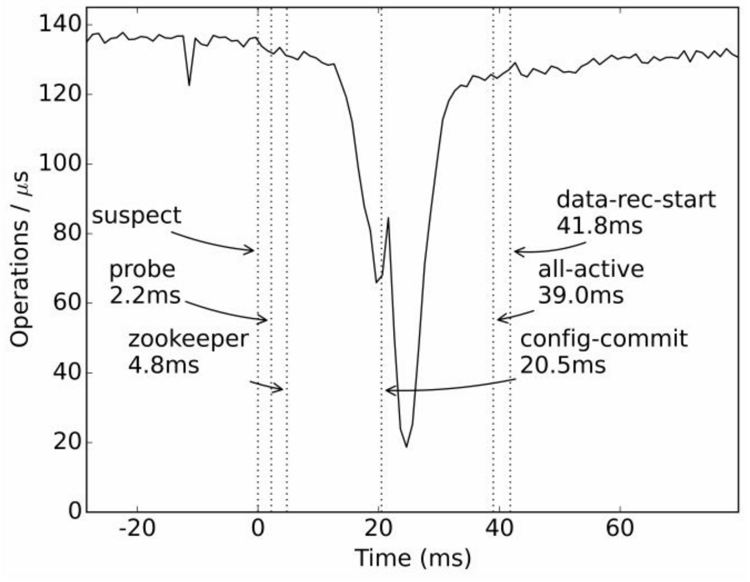
A: The RDMA writes has another challenge. How will server reject messages from old configurations? *Draining logs* ensures that all machines (backups) process all the records in their logs when they receive COMMIt message.

Q: **What is recovering transaction?**

A: A recovering transaction is one whose commit phase spans configuration changes, and for which **some replica of a written object**, **some primary of a read object**, or the coordinator has changed due to reconfig.

Evaluation

Cluster of 90 machines, with 3-way replication and each machine has 256 GB DRAM.



(a) Time to full throughput

Evaluation

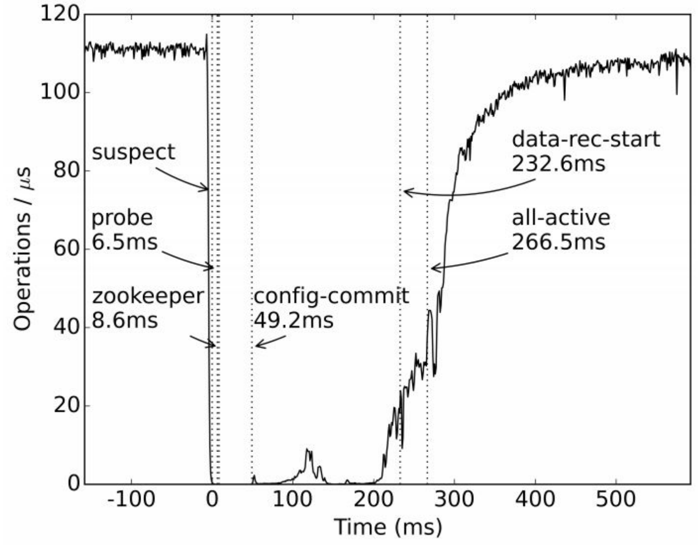


Figure 13. TATP throughput when failing 18 out of 90 machines at the same time

Summary

1. Super high speed distributed transactions
2. Hardware is exotic (NVRAM and RDMA) but may be common soon
3. use of OCC for speed and to allow fast one-sided RDMA reads

Next week

1. Let me know if you want to do final project 6.824 Project (mit.edu)
2. Watch lecture 15 (Big Data: Spark)
3. Read paper: zaharia-spark.pdf (mit.edu)
4. Start working on Lab 3b (hard!!)