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

Aleksandar Dragojevi´c, Dushyanth Narayanan, Edmund B. Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, Miguel Castro
Microsoft Research

Distributed transaction

Advantage:A very strong primitive abstract away concurrency and failures simplify building distributed systems.

Disadvantage: Not widely used because the poor performance and weak consistency.

Solution

FaRM, A main memory distributed computing platform can provide distributed transactions with strict serializability, high performance, durability, and high availability.

FaRM hit a peak throughput of 140 million TATP transactions per second and 90 machines with a 4.9TB database and it recovers from a failure in less than 50ms.

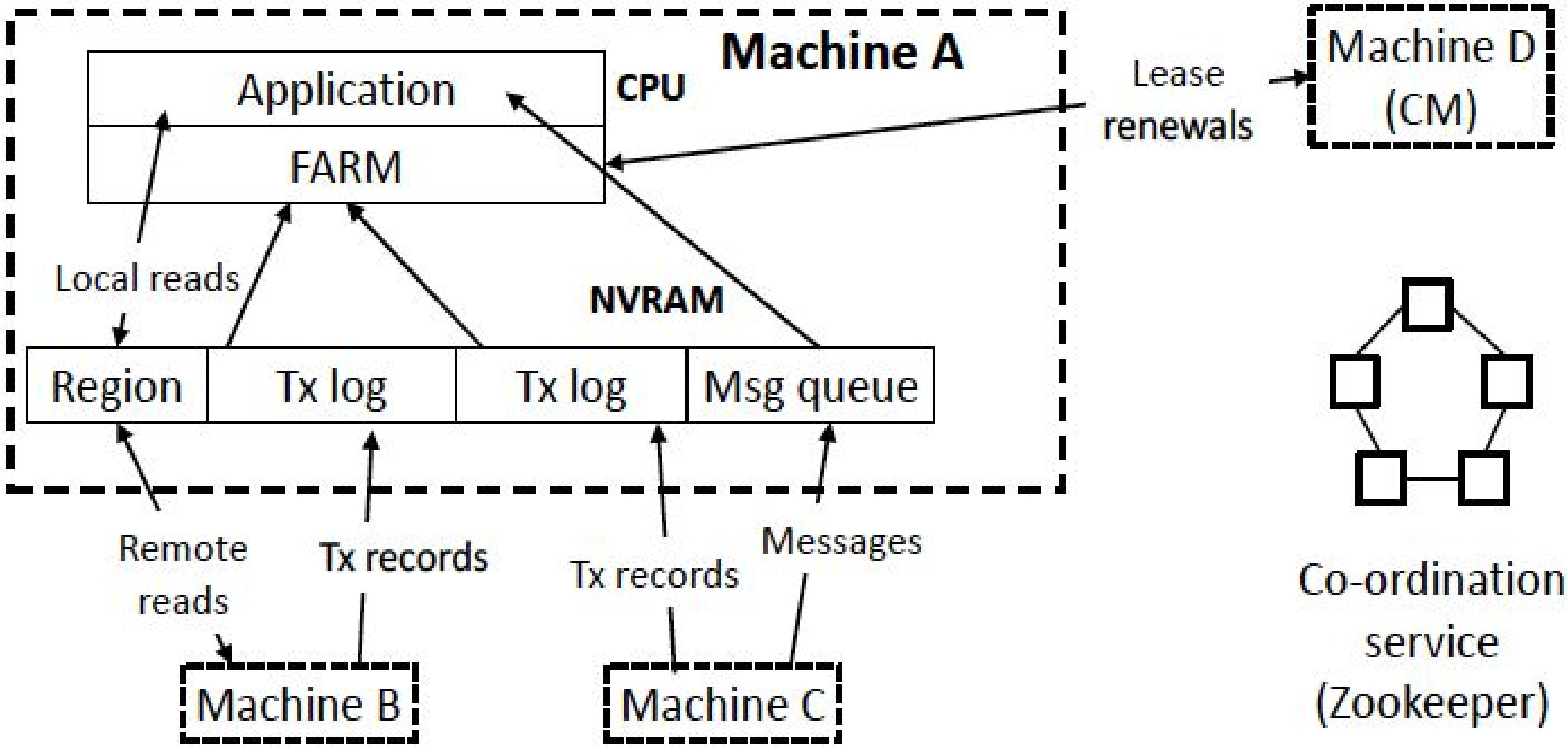
Hardware Trends

Non-volatile memory : Achieved by attaching batteries to power supply unit and writing the contents of DRAM (Cheap) to SSD when the power fails.

Fast Network with RDMA (remote direct memory access) which allows computer in a network to exchange data in main memory without involving the processor, cache or OS.

Optimize CPU overheads : reducing message counts, using one-sided RDMA reads and write, exploiting parallelism

Programming model and architecture


Machine A
Application
FARM
CPU
Local reads
NVRAM
Region
Tx log
Tx log
Msg queue
Remote reads
Tx records
Tx records
Messages
Machine B
Machine C
Lease renewals
Machine D (CM)
Co-ordination service (Zookeeper)

Overview

FaRM transactions uses optimistic concurrency control.

FaRM provides applications with the abstraction of a global address space among machines.

A FaRM instance moves through a sequence of configurations over time as machine fail or new machines are added.

Configuration

A configuration is a tuple <i, S, F, CM>

where i is a unique id, S is the set of machines in the configuration, F is a mapping from machines to failure domains, CM is the configuration manager.

Every machines in a FaRM instance agree on the current configuration and to store it.

Global address space and queues

The global address space consists of 2GB regions, each replicated on one primary and f backups.

Objects are always read from the primary copy using local memory accesses and using one-sided RMA reads if remote.

Each object has a 64-bit version that is used for concurrency control and replication.

FIFO queues are implemented for transaction or message queues.

Distributed transactions and replication

Brief Overview

1. Transaction protocols and replication protocols are integrated to improve performance.
2. Fewer messages than traditional protocols, and exploits one-sided RDMA reads and writes for CPU efficiency and low latency.
3. Primary-backup replication are stored in DRAM for data and transactions log.

Transactions protocol

Overview

FaRM guarantees that individual object reads are atomic, that they read only committed data, and reads of objects written by the transaction return the latest value written.

lock-free reads are provided, which are optimized single-object read only transactions.

During execution phase, transactions use one-sided RDMA to read objects and they buffer writes locally.

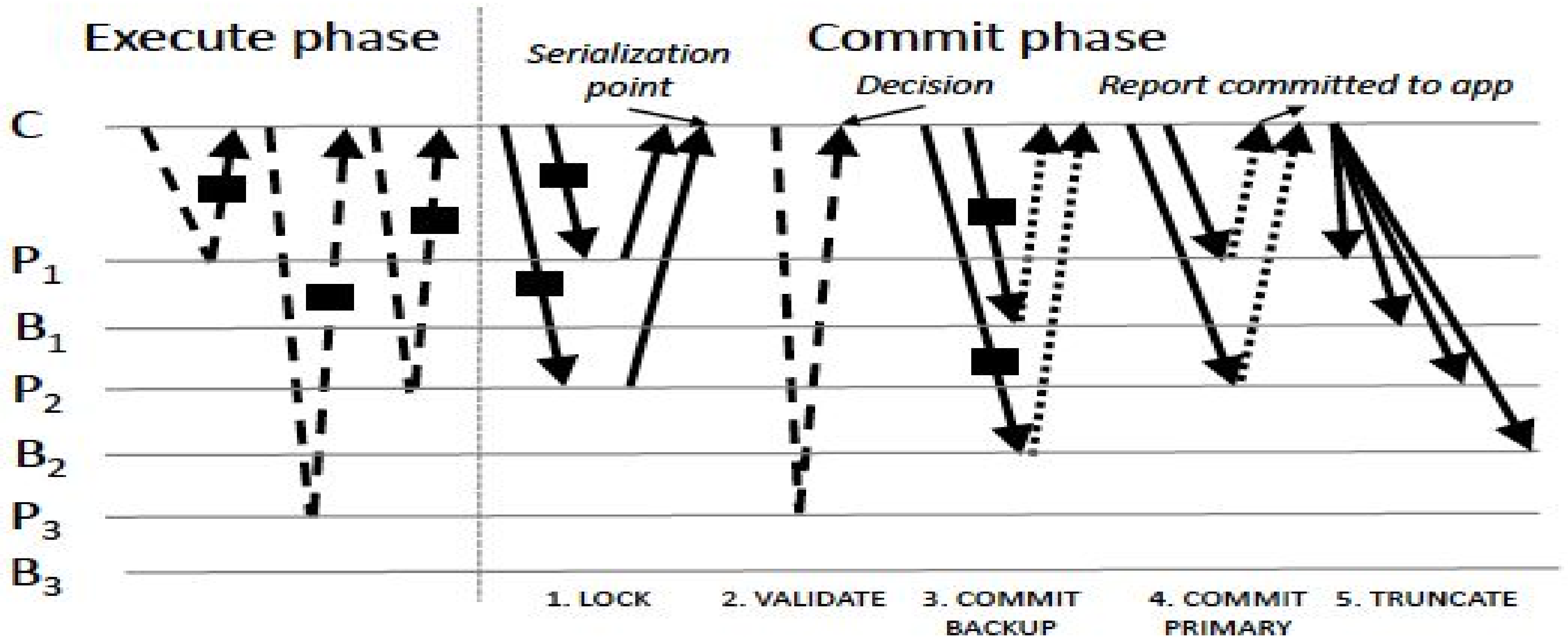


Figure 4. FaRM commit protocol with a coordinator C, primaries on P_1, P_2, P_3, and backups on B_1, B_2, B_3. P_1 and P_2 are read and written. P_3 is only read. We use dashed lines for RDMA reads, solid ones for RDMA writes, dotted ones for hardware acks, and rectangles for object data.

Lock and validation

Lock:

1. Coordinator writes a LOCK record to log on each machine that is a primary for any written object.
2. Primaries attempt to lock these objects at specified versions.
3. Coordinator aborts transaction and send abort record to all primaries if locking fail.

Validation:

1. Coordinator performs read validation (One-sided RDMA) by reading from their primaries and abort transaction if any object changed.
2. Use RPC if a primaries hold more than t_r objects.

Commit and Truncate

Commit:

1. The coordinator writes a COMMIT-BACKUP records to the non-volatile logs at each backup and waits for an ack from the NIC.
2. The coordinator writes a COMMIT-PRIMARY record to the logs at each primary if all backup acked. Then, Report to application if at least one of primaries acked.
3. Primaries update object and their versions, and unlocking them which exposes the writes committed by the transaction.

Truncate:

1. The coordinator truncates logs at primaries and backups after receiving acks from all primaries.
2. Backups update their copies of the objects at this phase.

Correctness

Serializability:

1. All the write locks were acquired.
2. Committed read-only transactions at the point of their last read.

Serializability across failures:

1. Waiting for hardware acks from all backups before writing COMMIT-PRIMARY.
2. Wait for at least one successful commit among primaries.

Failure recovery

Overview

1. FaRM provides durability and high availability using replication: all committed state can be recovered from regions and logs stored in non-volatile DRAM.
2. Durability is ensured even if at most f replicas per object lose the content of non-volatile DRAM.
3. Availability: Maintain availability with failures and network partition by provide a partition exists that contains a majority of the machines which remain connected to each other and to a majority of replicas in Zookeeper service, and the partition contains at least one replica of each object.

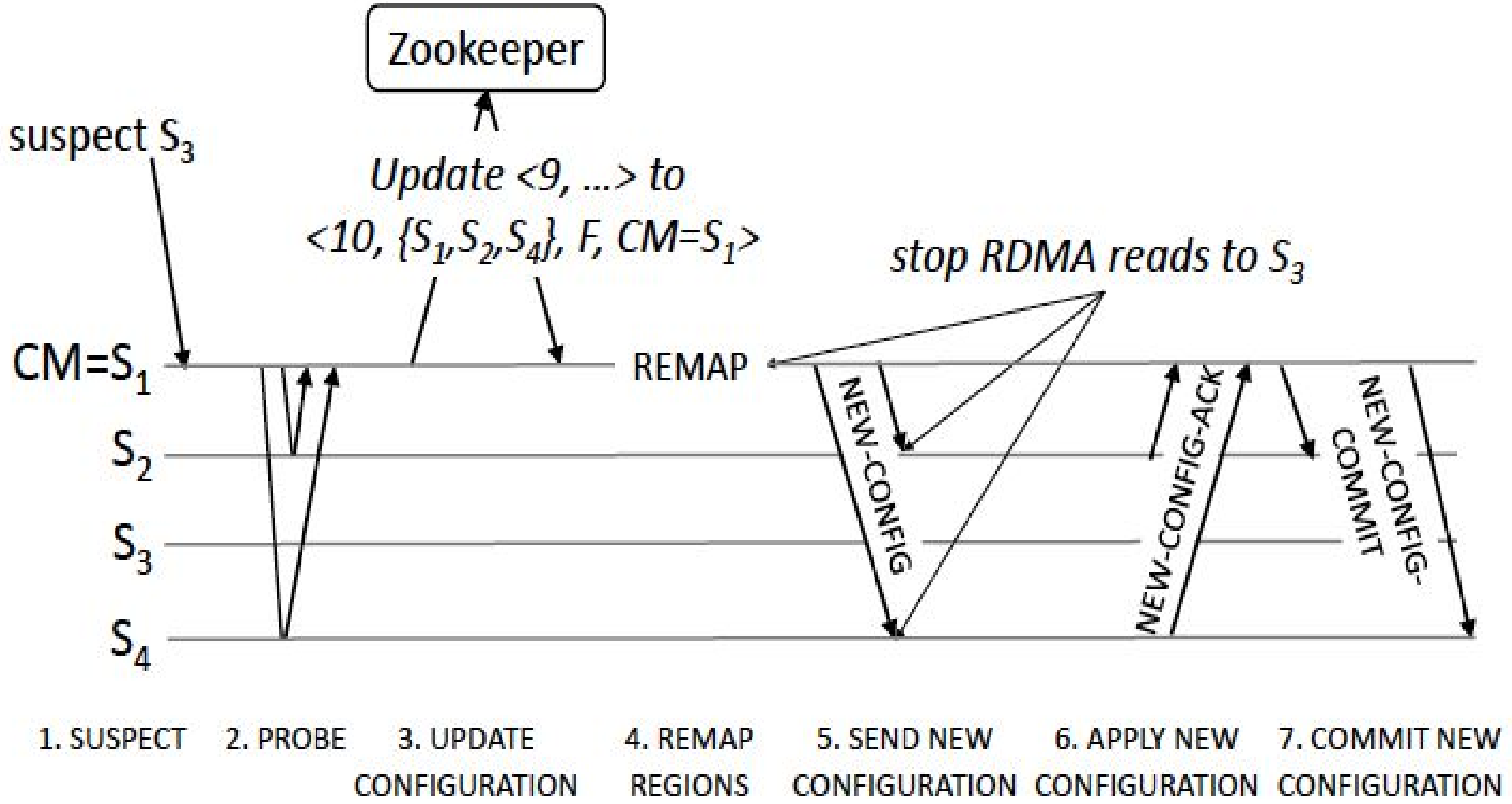
Failure detection

FaRM uses leases to detect failures.

1. Every machines holds a lease at the CM and the CM holds a lease at every other machine.
2. Expiry of any lease triggers failure recovery.
3. Each machine sends a lease request to the CM and it responds with a message that acts as both a lease grant to the machine and a lease request from the CM.

Reconfiguration

1. The reconfiguration protocol moves a FaRM instance from on configuration to the next.
2. One-sided RDMA operation require new protocol since the absence of remote CPU.
3. Precise membership: After a failure, all machines in a new configuration must agree on its membership before allowing object mutation. (7 Steps)


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

Timeline

Suspect:

1. When a lease for a machine expires at the CM, it suspects that machine of failure and initiates reconfiguration (block all external requests) and try to become the new CM.

Probe:

1. The new CM issue an RDMA read to all machines in the config, any machine for which the read fails is also suspected.(Correlated Failure)
2. The new CM proceed the reconfiguration after it obtains responses for a majority.

Timeline

Update configuration:

1. The new CM attempts to update the configuration data to $<c+1, S, F, CM_{id}>$.

Remap regions:

1. New CM reassigns regions mapped to failed machines to restore the number of replicas to f+1.
2. For failed primaries, a surviving backup is promoted to be the new primary for efficiency

Send new configuration:

1. New CM sends NEW-CONFIG message (config id, own id, id of other machines in this config) to all the machines in the configuration. Also, reset the lease protocol if CM has changed.

Timeline

Apply new configuration:

1. Each machine update it current configuration id after it receive the NEW-CONFIG message with a greater configuration identifier,
2. Each machine Allocates space to hold any new region replicas assigned to it, block all external requests and reply to the CM with a NEW-CONFIG-ACK message.

Commit new configuration:

1. After receiving ACKs back, new CM waits to ensure that any “old leases” expires, send NEW-CONFIG-COMMIT to all configuration members.
2. All configuration members can unblock previously blocked external requests.

Transaction state recovery

FaRM recovers transaction after a configuration change using the logs distributed across the replicas of objects modified by a transaction.

The timeline of transaction state recovery:

1. Block access to recovering regions.
2. Drain logs.
3. Find recovering transactions.
4. Lock recovery.
5. Replicate log records.
6. Vote
7. Decide

Transaction state recovery

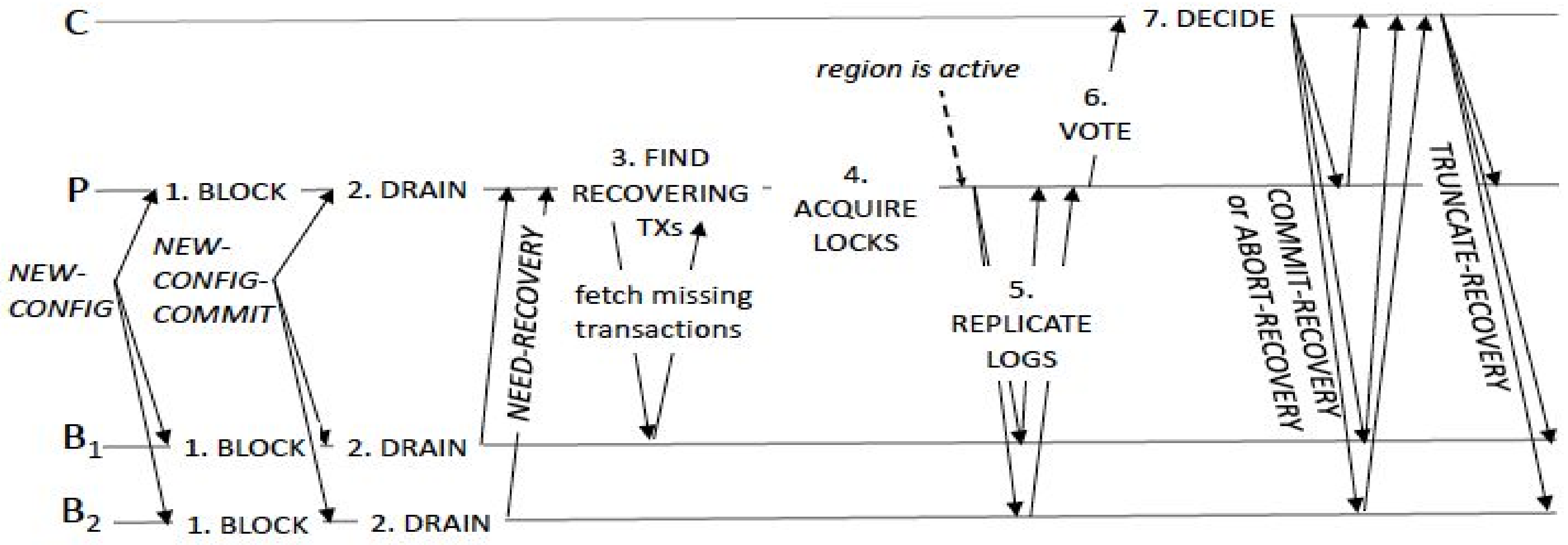


Figure 6. Transaction state recovery showing a coordinator C, primary P, and two backups B_1 and B_2

Step 1-2

Block access to recovering regions

1. Blocking requests for local pointer and RDMA references to the region until we finish lock recovery.

Drain logs

1. All machines process all the records in their logs when they receive a NEW-CONFIG-COMMIT.
2. They record the configuration identifier in a variable “LastDrained” when they are done.

Step 3-4

Find recovering transactions

1. Recovering transaction is one whose commit phase spans configuration changes, and for some replica of a written object, some primary of a read object, or the coordinator has changed due to reconfig.
2. Metadata is added during reconfiguration phase.
3. During log drain, the transaction id and list of updated region identifiers in each log record in each log examined the set of recovering transactions.
4. Each backup of a region sends NEED-RECOVERY msg to primary if needed.

Lock Recovery

1. The primary shards the transaction by id across its thread and lock any objects modified by recovering transactions in parallel.

Step 5-6

Replicate log records

1. The threads in the primary replicate log records by sending backups the REPLICATE -TXSTATE message for any transactions that they are missing.

Vote

1. The coordinator for a recovering transaction decides whether to commit or abort the transaction based on votes from each region updated by the transaction.
2. RECOVERY-VOTE msg is sent from primary to peer threads for each recovering transaction that modified the region
3. Corresponding msg are sent back to vote.

Step 7

Decide

1. Coordinator decides to commit if it receives a commit-primary vote from any region.
2. Otherwise, wait until there is at least one region voted commit-backup and all other regions modified by the transaction voted lock, commit backup, or truncated.
3. Otherwise, abort.

Correctness

Intuition: Recovery preserves the outcome for transactions that were previously committed or aborted.

1. Step 2 ensures a log record for a recovering transaction that committed is guaranteed to be processed and accepted before or during log draining.
2. Step 3-4 ensures that the primaries for the region modified by the transaction see these records.
3. Records are replicated to the backups to guarantee that voting will produce same results.
4. Primaries send votes to coordinator based on the records they have seen.
5. Decide step guarantees that the coordinator decides to commit any transaction that were committed before.

Recovering data

1. FaRM must recover data at new backups for a region to ensure that it can tolerate f replica failures in the future.
2. Each machine sends REGIONS-ACTIVE msg to CM when all regions for which it is primary become active. The CM sends a ALL-REGIONS-ACTIVE to all machines in the config after receiving all those msg. FaRM begins data recovery for new backups.
3. Recovered object need to be examined before being copied to the backup.(Version Comparison).

Recovering allocator state

1. FaRM allocator splits region into blocks(1MB) and keeps metadata called block headers.
2. Block header consists object size and slab free list and replicated to backups when a new block is allocated. (Ensure availability after failure)
3. After a failure, slab free lists are recovered on the new primary by scanning objects in the region.

Evaluation

Setup

Testbed: 90 machines used for a FaRM and 5 machines for a replicated ZooKeeper.

Machine spec: 256GB of DRAM and two 8-core Intel E5-2650 CPUs running Window Server 2012 R2.

Hyper-threading are enabled and the first 30 threads for the foreground work and the remaining 2 threads for the lease manager.

FaRM Config: 1 primary and 2 backups with a lease time of 10ms.

Benchmarks

TATP (Telecommunication Application Transaction Processing):

1. %70 of the operations are single-row lookup by using FaRM's lock free reads (no commit phase).
2. %10 of the operations read 2-4 rows and require validation during the commit phase.
3. %20 of the operations are updates and require full commit protocol.

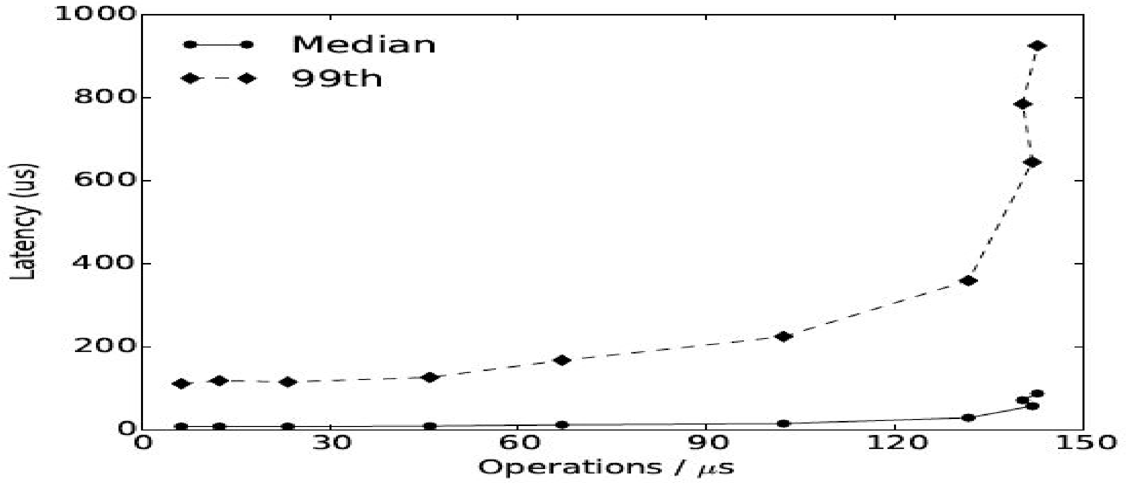
TPCC, a well known benchmark with complex transactions that access hundreds of rows.

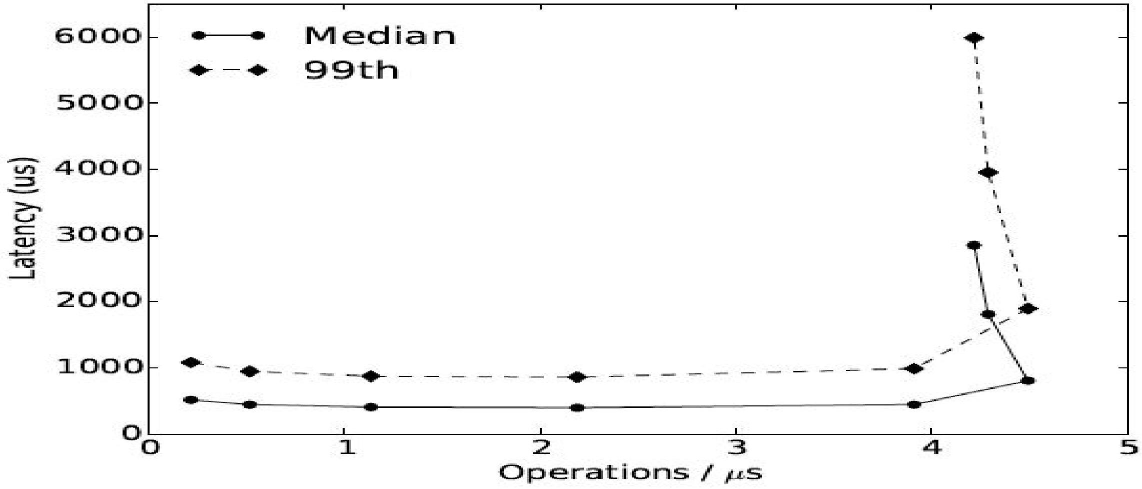
Normal case performance(Failure-free)

TATP: 140 million TATP transaction per second with 58 us median latency and 645 us 99th percentile latency. (Outperforms published TATP results for Hekaton)

TPCC: Forms up 4.5 million “new” transaction per second with median latency of 808 us and 99th percentile latency of 1.9ms.

(Throughput is 17x higher than Silo without logging)

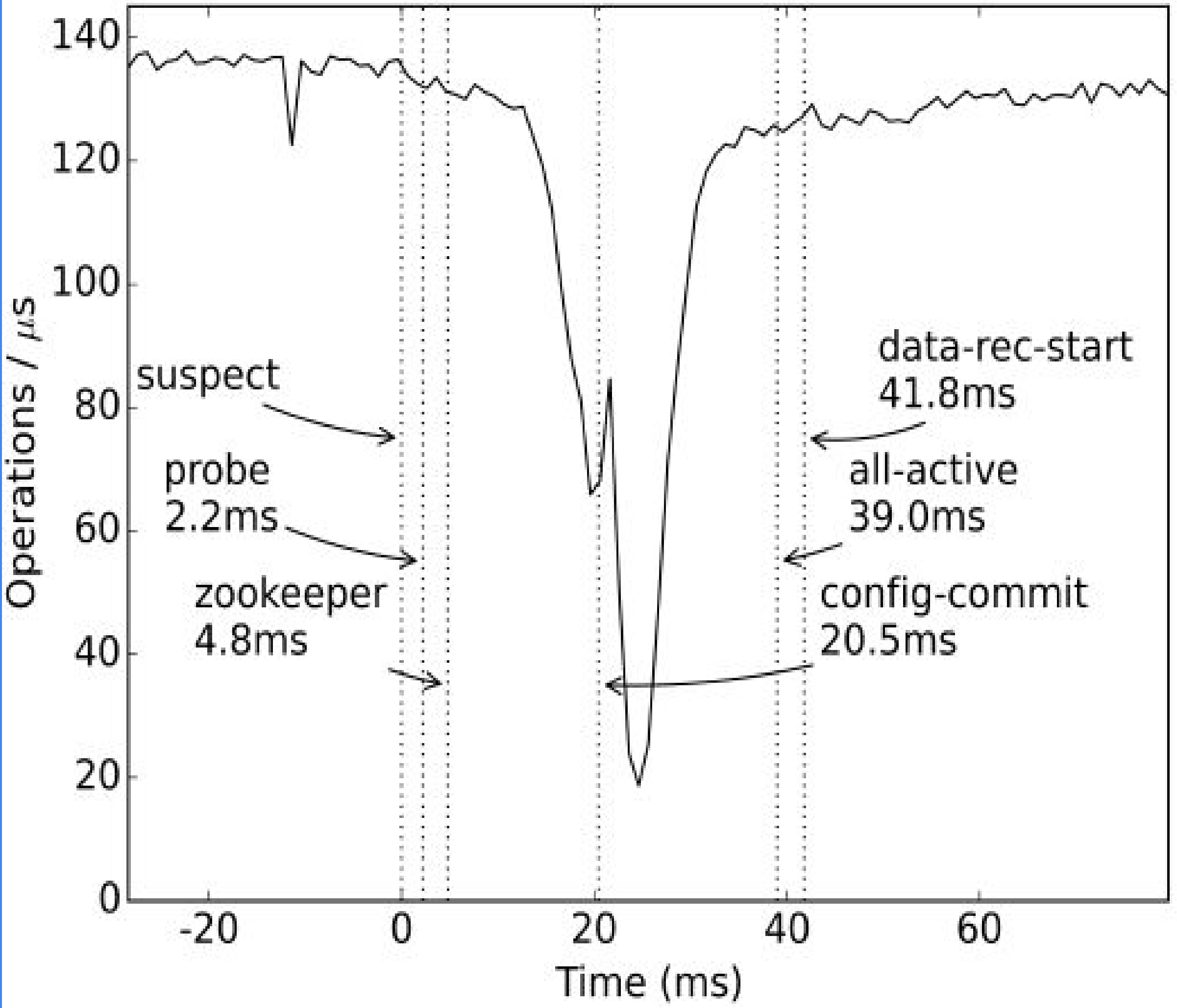




Failures

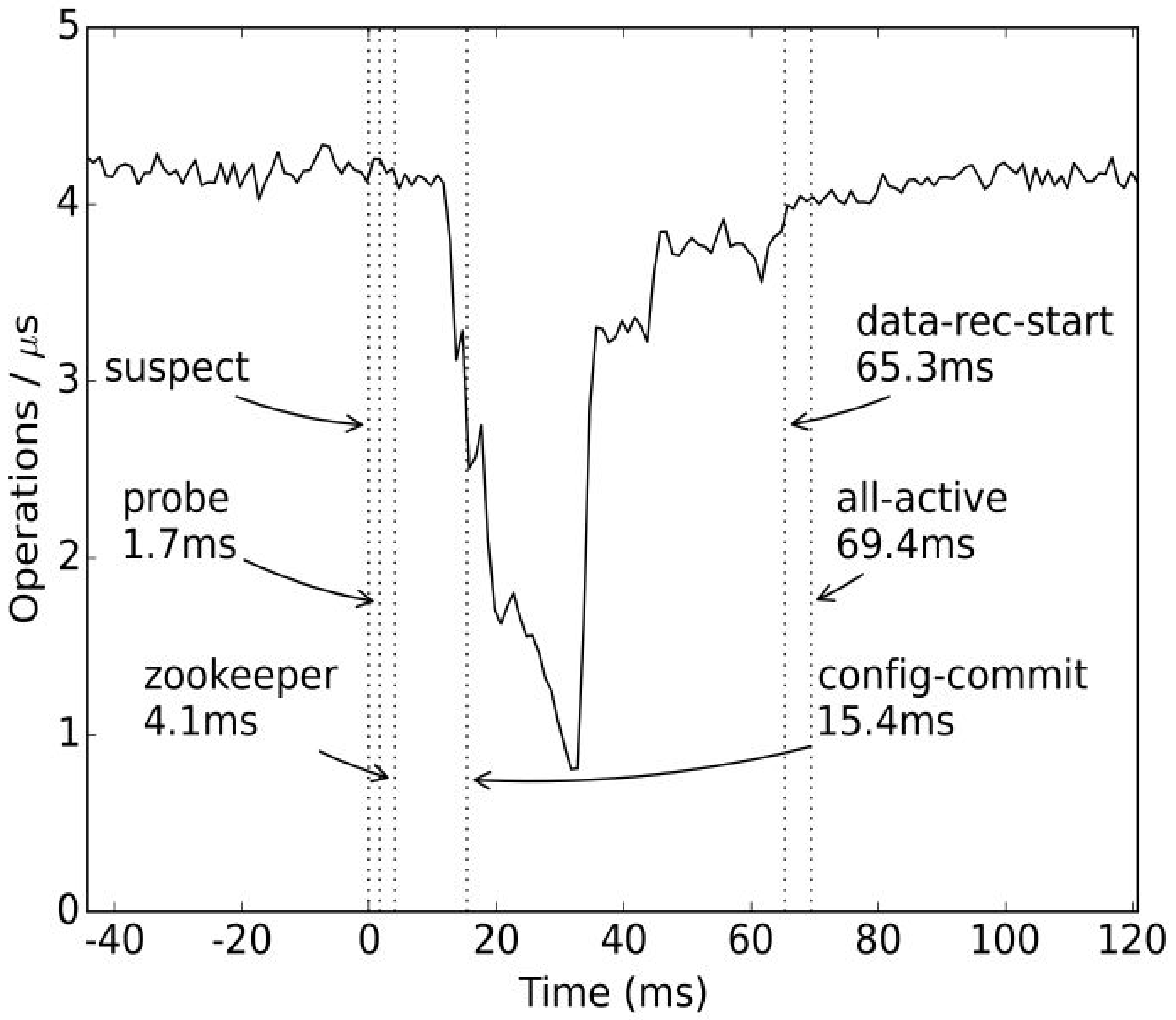
TATP: Throughput drops sharply at the failure but recover rapidly(40ms) and have some dips because of skewed access in the benchmark.

TPCC: Regain most of throughput in less than 50ms and that all regions become active shortly after that.



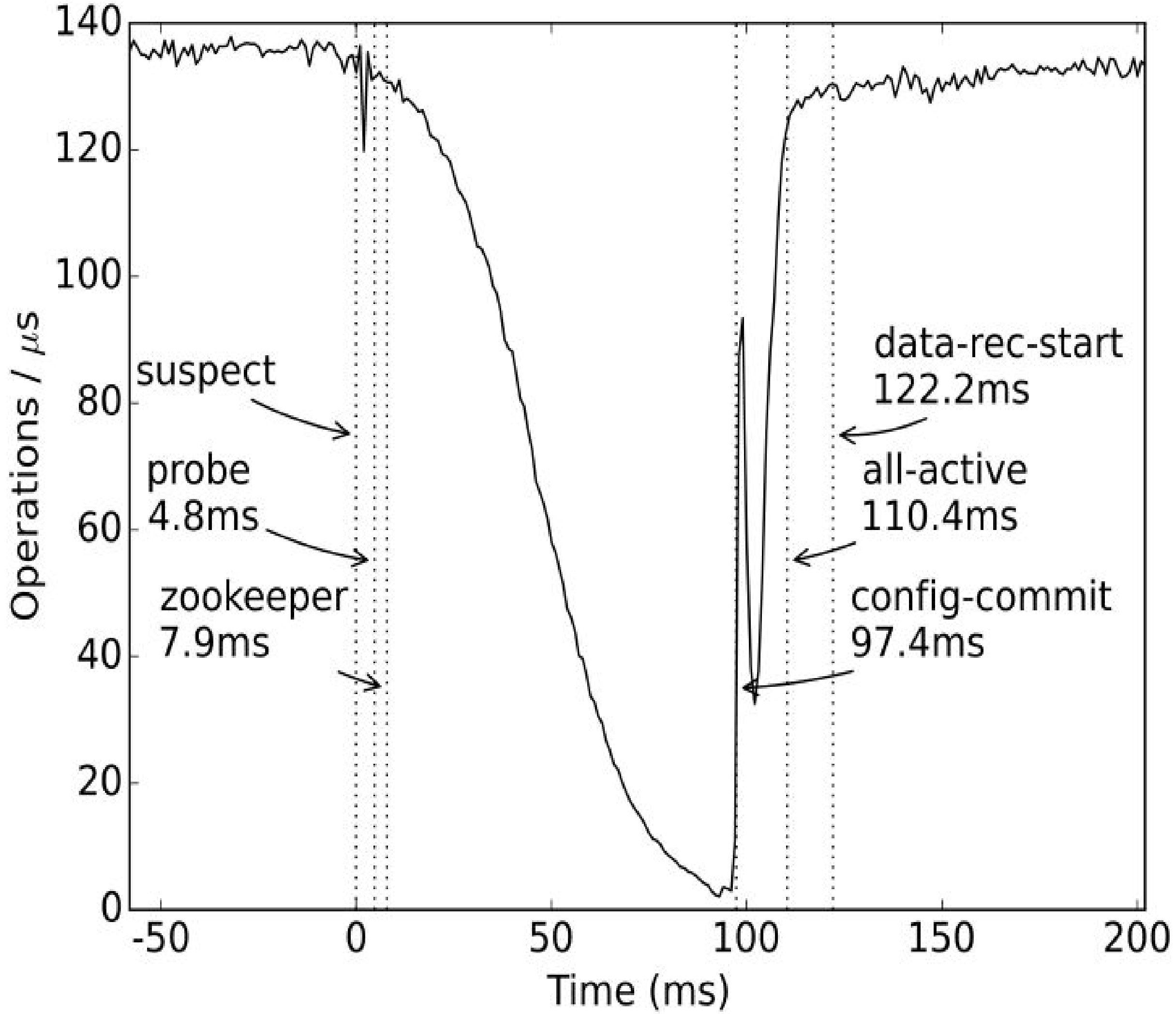
Failures

TPCC: Regain most of throughput in less than 50ms and that all regions become active shortly after that.



Failing the CM

Recovery is slower than when a non-CM process fails because the increasing reconfiguration time: from 20 ms to 97 ms.

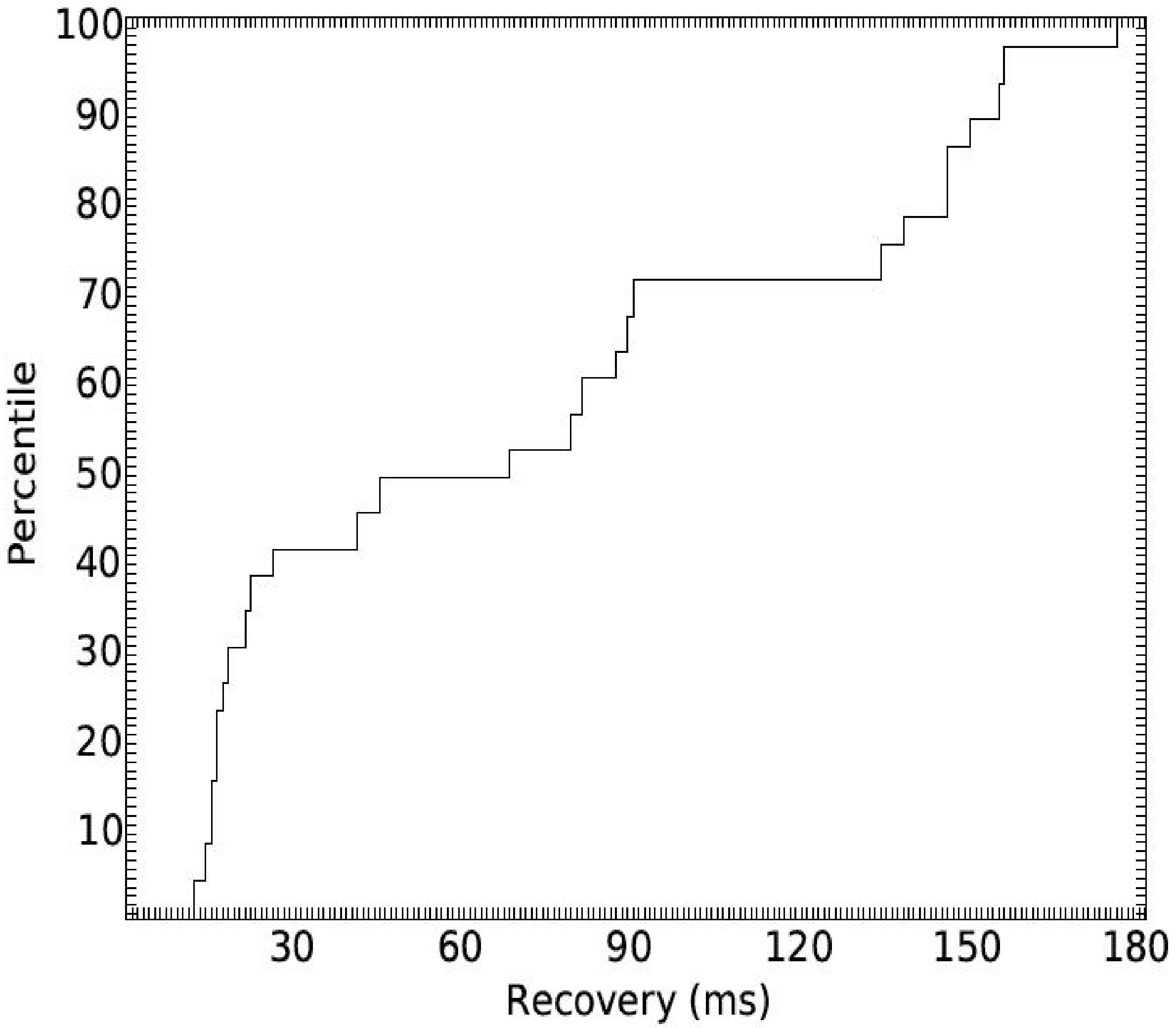


Distribution of recovery times

Same process time to regain throughput when use smaller data set.

Median recovery time: Around 50 ms

More than %70 executions have recovery time less than 100 ms.



Related work

1. RAMCloud: Does not support multi-object transaction where FaRM supports, takes too long for recovery where FaRM just use tens of a milliseconds of a failure.
2. Spanner: Use 2f+1 replicas compared to FaRM's f+1, and sends more messages to commit than FaRM.
3. Silo: Storage is local and thus availability is lost when the machine fails, FaRM also achieves higher throughput and lower latency than Silo.

Conclusion

1. Transaction make it easier to program distributed systems but many systems avoid them or weaker their consistency to improve availability and performance.
2. FaRM is a distributed main memory computing platform for modern data centers that provides strictly serializable transactions with high throughput, low latency, and high availability.
3. FaRM can also recover from a machine failure back to providing peak throughput in less than 50 ms , making failures transparent to applications.