No Compromises

Distributed Transactions with Consistency, Availability, Performance

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

Why

Transactions with high availability & strict serialization

- Abstraction: endlessly-running single machine with one transaction at a time with consistent ordering
- Issue: Poor Performance
    - No Transactions, Weak Consistency, Single Machine Transactions,

What

Distributed ACID transactions with:

- Strict Serializability
- High Availability
- High Throughput
- Low Latency

What - Performance

FaRM

- Peak throughput: 140 million TATP transactions/sec
- 90 machines
- 4.9 TB database
- 50< ms Recovery

How

RMDA

cheap non-volatile DRAM

How - Design

1) Reduce Message Counts
2) RDMA Reads/Writes
3) Parallelism

Assumptions: Network is fast, we are cpu-bound

How - Design

Assumptions:

- Network is fast, we are cpu-bound

Hardware-DRAM

Distributed UPS means durable DRAM -> SSD rarely > NVDIMMs

- Energy-bound

Hardware-RDMA

- No Remote CPU
    - > 2x RPC / > 4x RPC (after optimization)
    - Bottleneck: NIC Message Rate
    - After optimizations: CPU Bound

Software - Programming Model

Software - Programming Model

- Global Address Space (2GB Regions, 1 primary, *f backups*)
  - CM Manages config with monotonically increasing counter & chooses replicas (magic)
  - 2PC to make new region
- Local & Remote Objects in Transactions
- Transactions (OCC)
  - App Thread can start
  - Said thread becomes coordinator
  - Thread can execute arbitrary logic - e.g. read/write/allocate/free objects
  - Calls FaRM to commit at end using 4 stage commit
  - *Strict Serialiability*

Software - Programming Model

- Transactions (OCC)
    - Individual Object Reads atomic
    - only read committed data
    - successive reads return same data
    - reads of objects written *in same transaction* guarantee that value read as well
    - *no atomicity for reads across objects, but guarantees will not commit on issue? (commit time consistency checks)*
- Locality Hints
- Lock-free Reads

Software - Config

- Zookeeper to agree & store current config
- Custom RDMA implementation for:
    - Leases, Failure Detection, Coordinate Recovery
- Per machine FIFO ring-buffer transaction log (RDMA) & message queue
    - Lazily updates receiver

Software - Transaction & Replication

(Novel Part here)

- Fewer Messages
- RDMA
- Primary Backup (vertical paxos) for data/transaction logs
- *unreplicated transaction coordinators*

Software - Transaction & Replication

1) Lock
2) Validate
3) Commit Backup
4) Commit Primary
5) Truncate

Software - Transaction & Replication

1) Lock
   - Write LOCK record for written object primaries (versions & values) using CAS
   - Fails, Writes abort record to all primaries
   - Returns app error

Software - Transaction & Replication

1) Lock
2) Validate
   - Fail if any version # not expected by transaction
   - Abort
   - RDMA/RPC Reads if > 4 objects

Software - Transaction & Replication

1) Lock
2) Validate
3) Commit Backup
   - Write COMMIT-BACKUP record to backup logs (same as LOCK record contents)
   - Waits for NIC ack(s)

Software - Transaction & Replication

1) Lock
2) Validate
3) Commit Backup
4) Commit Primary
   - Coordinator writes COMMIT-PRIMARY record to logs at primaries
   - Success = >= 1 ACK/Local Write
   - In place objects update, version update, unlock object (exposing new vals)

Software - Transaction & Replication

1) Lock
2) Validate
3) Commit Backup
4) Commit Primary
5) Truncate
  - Backups/Primaries keep records till truncated
  - Coordinator truncates primaries/backups after *all* primaries send ACKs
  - Transactions to truncate piggy back in other log records,
  - Backups apply updates at truncation

Software - Transaction & Replication Correctness

- Read-Write Transactions
  - Serializable at write locks
- Read Transactions
  - Serializable at last read
- Strict Serializability
  - serialization point: Start of execution
  - completion reported to application

Software - 4 PC differences

- Coordinator reserves log space for PC records before starting in participants log’s
- f+1 replicas vs 2f+1 replicas
    - 4*Participants*(2f+1) messages > Participant writers*(f+3) RDMA writes / Participant readers*1 only

Software - Failure Recovery

(novel)

- Bounded Clock Drift / Eventually Bounded Message Delays
- DRAM for transaction log durability
- 5 phases

Software - Failure Recovery

1) Failure Detection
2) Reconfig
3) Transaction State Recovery
4) Bulk Data Recovery
5) Allocator State Recovery

Software - Failure Recovery

1) Failure Detection
   - Leases using 3 way handshake
     - magic: 5ms lease time - super short leases
   - Dedicated infiniband send/receive verbs
   - Dedicated Queues
   - Highest CPU scheduling priority & manager
   - Dedicated HW threads
   - Preallocated memory, paged & pinned

Software - Failure Recovery

1) Failure Detection
2) Reconfig
 - Leases (usual solution, but not here)
 - *Precise Membership*
   - After failure, agreement on membership before mutation
 - suspect, probe, update configs (after majority), remap regions, send new config, apply new config, commit new config

Software - Failure Recovery

1) Failure Detection
2) Reconfig
3) Transaction State Recovery
   - Block Region Access
   - Drain all message logs
   - Find recovering transactions
   - Build complete set of recovering transactions
   - Multithreaded recovery
   - Send recovered records to backups
   - Vote to determine whether to commit/abort transaction where coordinator through Consistent Hashing
   - Decide

Software - Failure Recovery

1) Failure Detection
2) Reconfig
3) Transaction State Recovery
4) Bulk Data Recovery
- Delay recovery till all regions ative and then copy in background in parallel
- 8 KB blocks
- Recovery reads start within random interval of previous read, examine before copy

Software - Failure Recovery

1) Failure Detection
2) Reconfig
3) Transaction State Recovery
4) Bulk Data Recovery
5) Allocator State Recovery
   - Regions are 1 MB blocks
   - Block Headers (replicated)
   - Slab Free Lists (recovered using a scan)
     - Backgrounded

Performance Graphs

(on back of paper)