Transactions on RAMCloud

SEDCL Forum

January, 2015

Collin Lee and Seo Jin Park
Stanford University

Overview

- Goals
- API & Semantics
- Commit Protocol
- Recovery Protocol
- Implementation Details
- Using Linearizability
- Conclusion
- Questions and Feedback

Transactions Goals

What are we trying to build?

- **Multi-object atomic updates**
- **Tolerate client failures**
- **Performance**
  - Low-latency
  - Large scale: 1M+ clients
- **Simple programmer interface**
- **Non-goals and assumption:**
  - No long running transactions
  - Small commit sets: 100 objects or less

Transaction Client API

```
class Transaction {
        read(tableId, key) => blob
        write(tableId, key, blob)
        delete(tableId, key)
        commit() => COMMIT or ABORT
}
```

- **Optimistic concurrency control**
- **Client-side transaction cache**

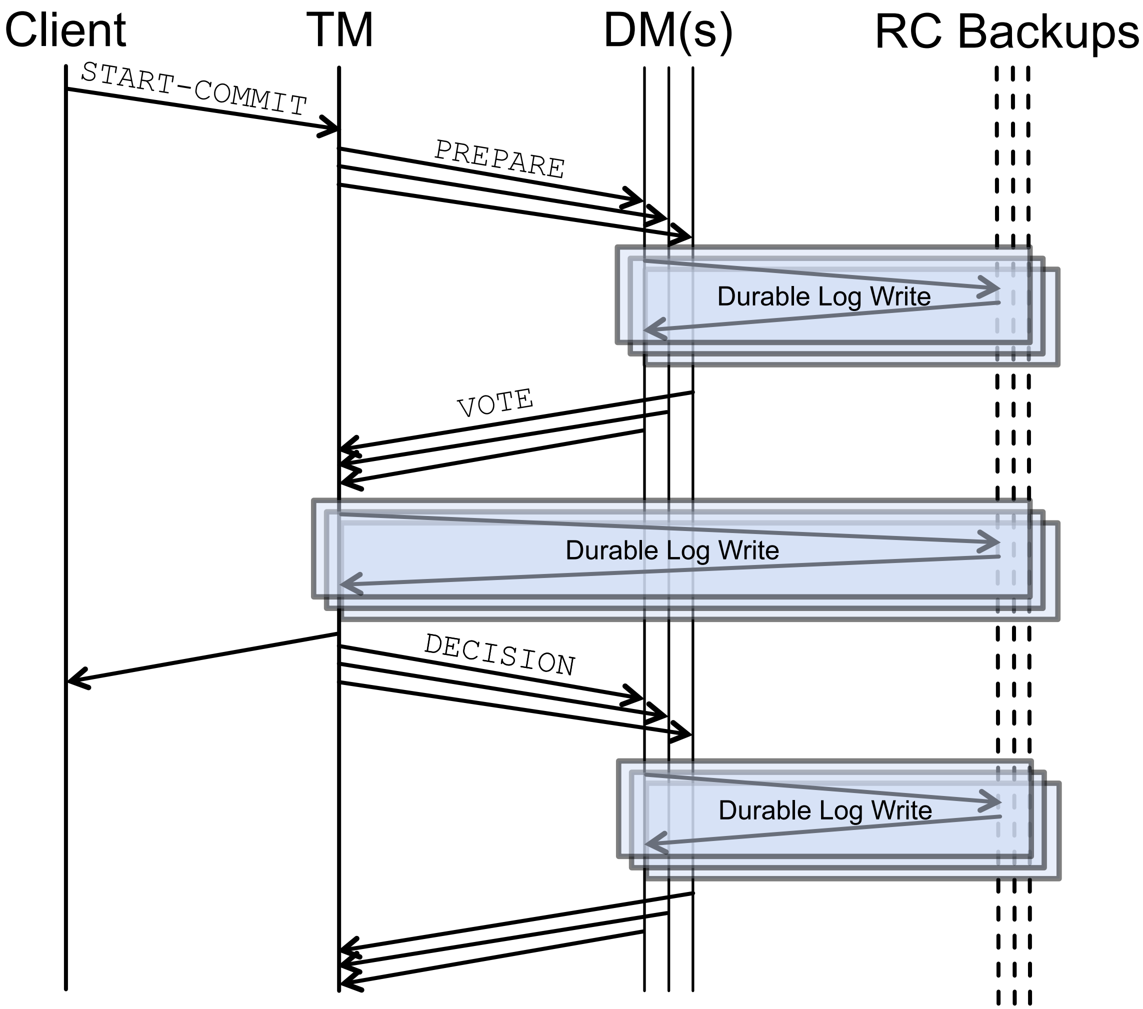
Transaction Commit Semantics

- **Multi-object conditional operation**
  - Operations are conditioned on a version
  - Commit succeeds if all operation conditions are met

```
ReadOp {tableId, key, version}

WriteOp {tableId, key, version, blob}

DeleteOp {tableId, key, version}


commit(OpList[]) => COMMIT or ABORT
```

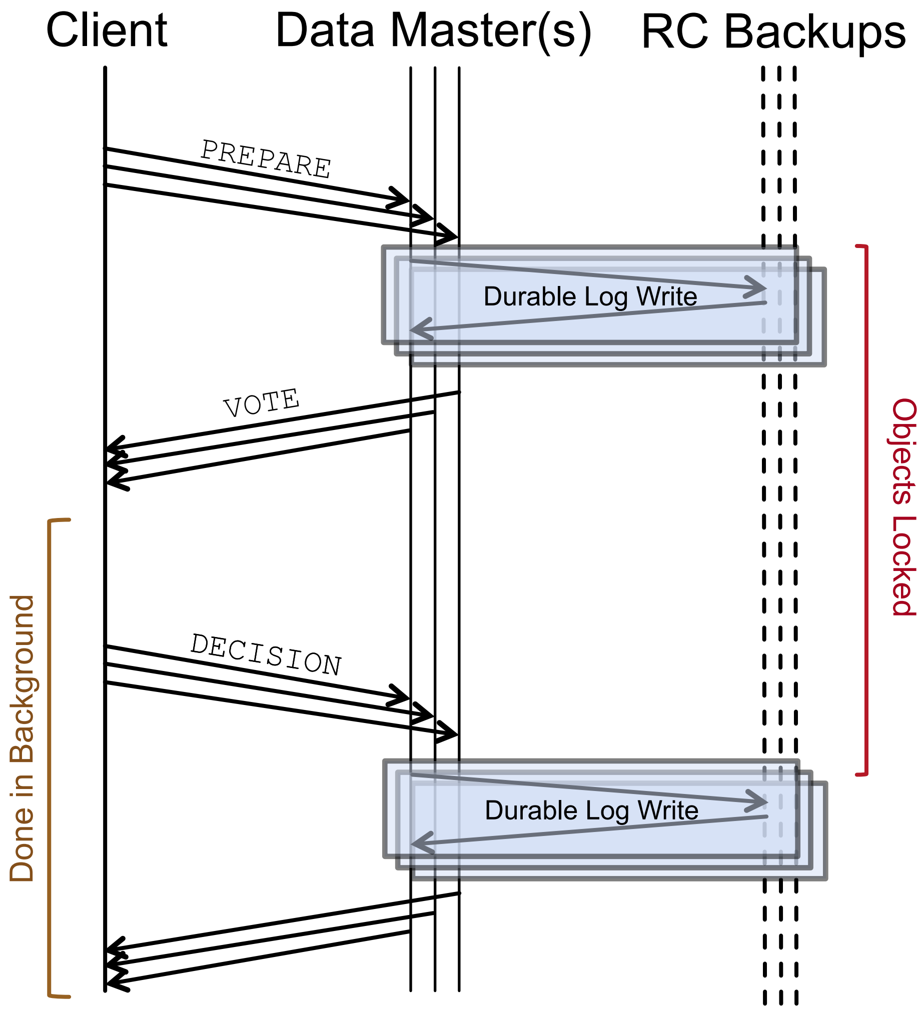
Transaction Commit (Attempt 1)

- **Standard 2PC with remote Tx Manager**
- **2 RTT + 2D (1.5 RTT + 1D with optimization)**
- **Other systems do better**
- **Can we leverage linearizability?**



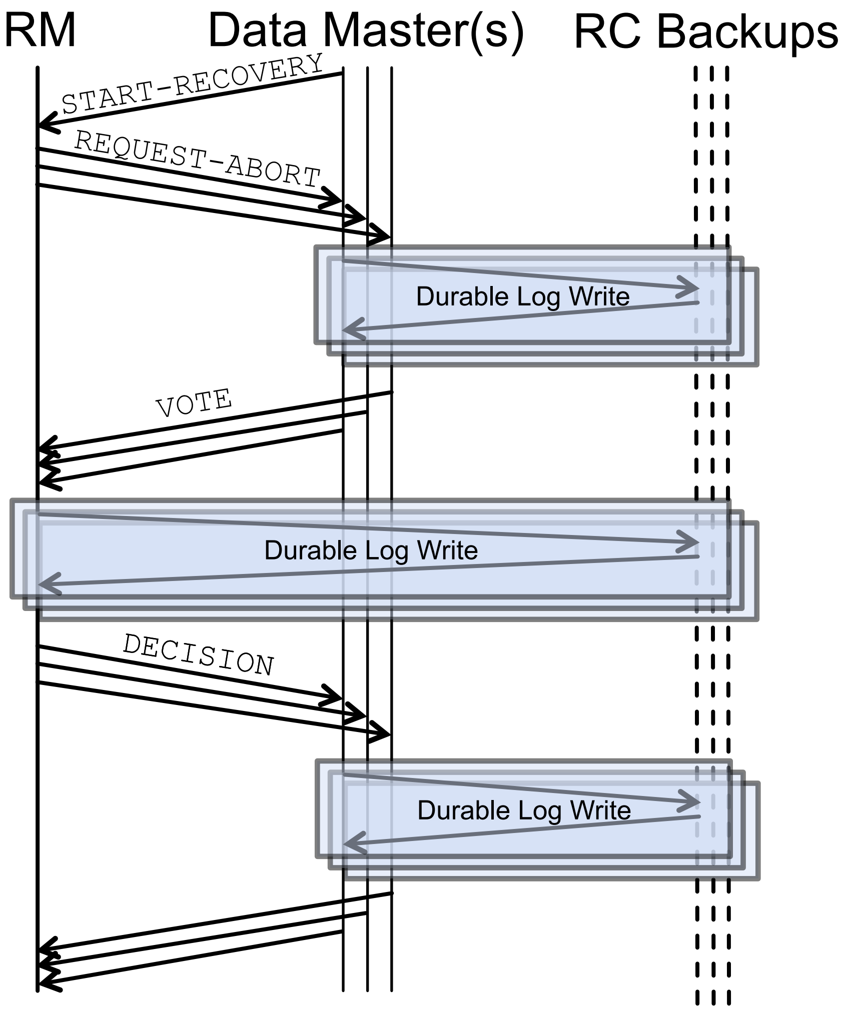
Transaction Commit

- **Client driven 2PC**
- **RPCs:**
  - `PREPARE() => VOTE`
  - `DECISION()`
- **Client blocked time: 1RTT + 1D**
- **Decisions sent in the background**
- **Better normal case operation**



Transaction Recovery

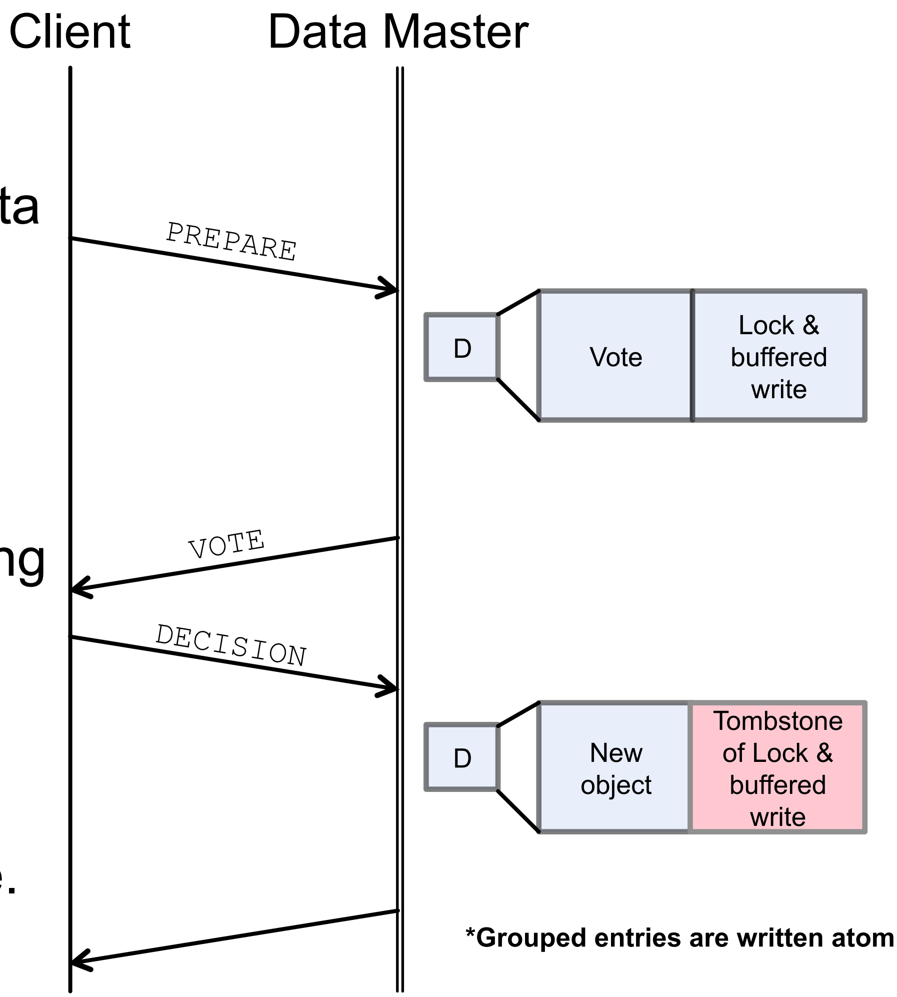
- **Server driven recovery**
- **RPCs:**
  - `START-RECOVERY()`
  - `REQUEST-ABORT() => VOTE`
  - `DECISION()`
- **Initiated by "worried" data masters**
- **Reuses common infrastructure**



What is Durably Logged?

Two types of log data

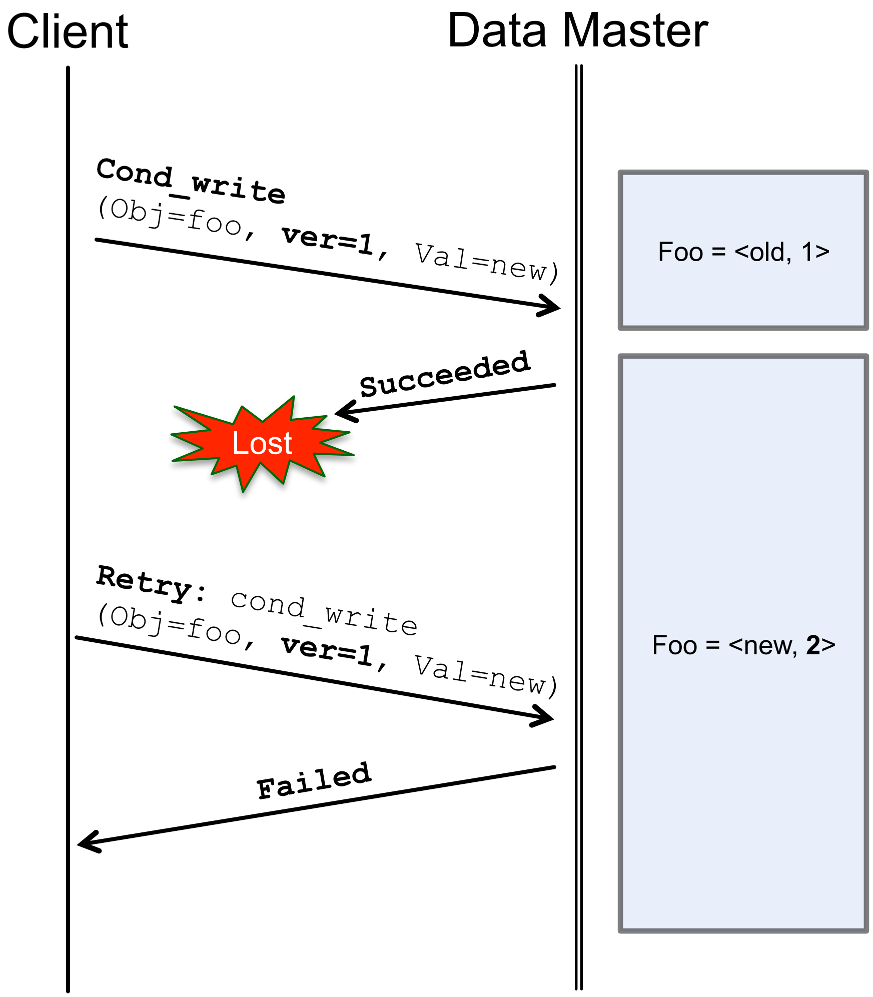
- **Lock & buffered write**
  - During crash recovery of data master, the lock will be grabbed again.
  - The buffered new value will be written on **COMMIT**.
  - This record is removed during processing **DECISION**.
- **Vote**
  - Automatically managed by **linearizability** infrastructure.



***Grouped entries are written atomically.**

Linearizable RPC Revisited (1)

- **RPCs in RAMCloud were not linearizable.**
    - If response of RPC is lost (either on network layer or on data master’s crash), client doesn’t know RPC was processed or not.
    - If client retries same RPC, it may see inconsistent result.


Client

Data Master

Cond_write
(Obj=foo, **ver=1**, Val=new)

Foo = <old, 1>

Succeeded

Lost

Retry: cond_write
(Obj=foo, **ver=1**, Val=new)

Foo = <new, **2**>

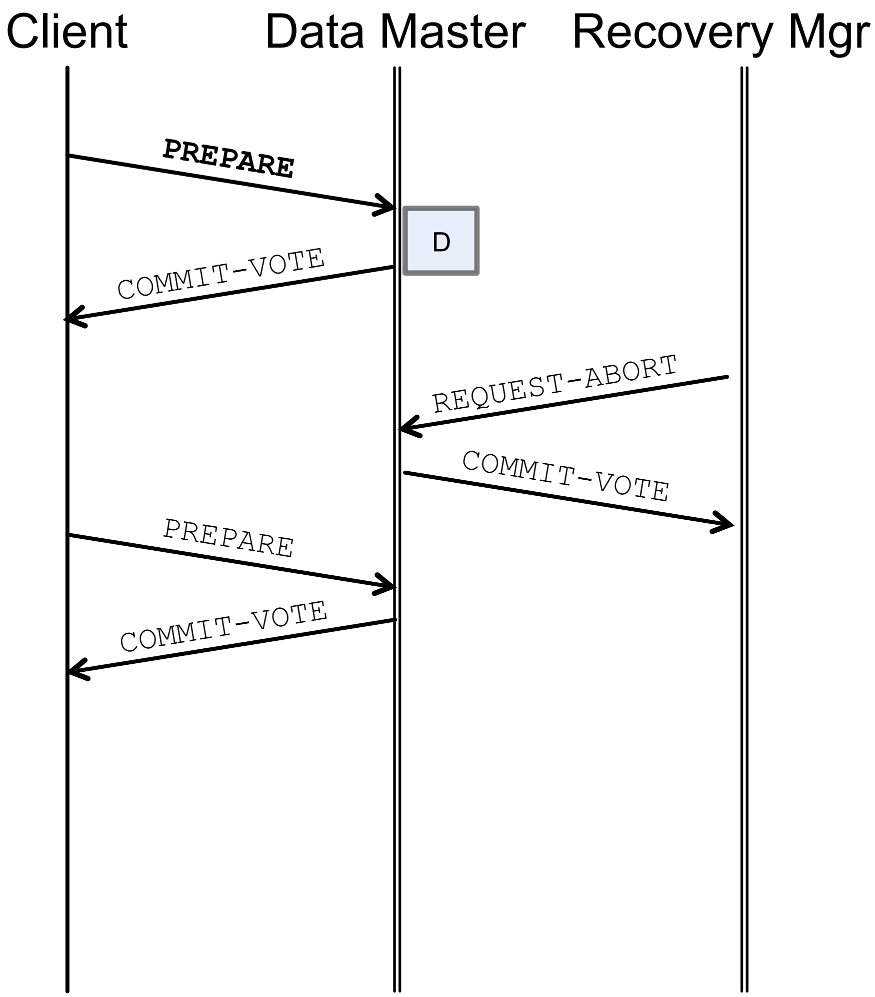
Failed

Linearizable RPC Revisited (2)

- **Save the results** of RPCs in log durably **until client acknowledges** its receipt, and respond with the old result **without re-executing**.
  - Lightweight solution on RAMCloud
  - Write latency penalty: ~200ns.
  - Per client state is 170 bytes and each server can sustain 1M clients with 170 MB of storage overhead.

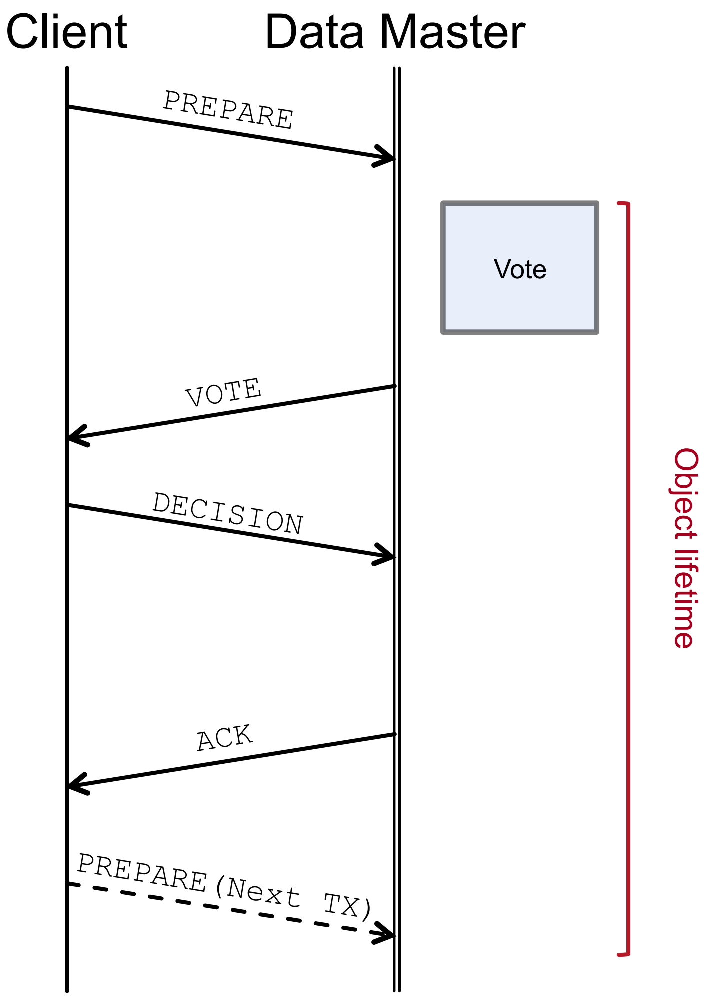
Linearizability Simplifies TX Recovery

- **Vote is saved durably by linearizability infrastructure.**
  - Since response, **VOTE**, is saved, we always get same vote for any retried **PREPARE**.
  - Crash recovery of data master can be handled by resending **PREPARE**.
  - On client crash, transaction recovery coordinator resend **PREPARE** as if it was sent from client **(with request to abort if possible).**



Linearizability Simplifies GC

- **Challenge: when to delete record of vote & free up space?**
  - Client knows outcome
  - All servers applied DECISION durably.
- **Sinfonia: all-to-all msg of applied TX IDs.**
- **Leverage linearizability:**
  - As client acknowledges the completion of TX, servers may remove vote log.



Conclusion

- **Client-driven multi-object atomic updates in 1RTT + 1D synchronous time**
- **Simplifying transactions using linearizability**
- **Progress**
  - Linearizability implementation DONE!
  - Transactions implementation (early next month)
- **What's next?**
  - Looking for feedback
  - Benchmarking & Systems for comparison
  - API improvements & performance tuning