10 APPENDIX

10.1 Proof of Linearizability in FDBKeeper

Theorem 10.1 (Linearizability of FDBKeeper). *FDBKeeper guarantees the linearizability of operations by combining the strict serializability of FDB with client-side locking schemes.*

Proof. We prove this theorem by constructing a global linear history and demonstrating that it satisfies the properties of linearizability. This proof formalizes FDBKeeper's three-type client-side locking mechanism and dependency-based execution model, showing how it implements the same two ordering guarantees as ZooKeeper: cross-client linearizability and FIFO client order.

10.2 Notation and Definitions

Basic Sets and Elements.

- O: Set of all operations
- C: Set of all clients
- $\mathcal{P}$: Set of all paths (e.g., ZNode paths)
- $Values$: Set of all possible data values

Operation Properties.

- $o, o_i, o_j \in O$: Operations
- $T_s(o), T_e(o) \in \mathbb{N}$: Start and end times of operation o, measured by a global clock
- $\tau(o) \in \{\text{WRITE}, \text{READ}\}$: Type of operation o
- $\pi(o) \in \mathcal{P}$: Path on which operation o operates
- $\gamma(o) \in C$: Client that issues operation o
- $val(o) \in Values$: Value written by operation o if $\tau(o) = \text{WRITE}$; undefined if $\tau(o) = \text{READ}$
- $result(o) \in Values$: Result returned by operation o

Path Relationships.

- $ancestors(p) \subset \mathcal{P}$: Set of all ancestor paths of path p (e.g., $ancestors(\text{“/a/b”}) = \{\text{“/”}, \text{“/a”}\}$)

Order Relations.

- $<_{rt}$: Real-time partial order, where $o_i <_{rt} o_j$ if $T_e(o_i) < T_s(o_j)$ (i.e., o_i completes before o_j begins)
- $<_c$: Client FIFO partial order, where $o_i <_c o_j$ if o_i and o_j are issued by the same client and o_i is issued before o_j
- $<_H$: Total order constructed by FDBKeeper, represented as $H = \langle o_1, o_2, \ldots, o_n \rangle$, where $o_i <_H o_j$ if $i < j$
- $S = \langle t_1, t_2, \ldots, t_m \rangle$: FDB transaction sequence, where t_i represents a transaction
- $\sigma : O \to S$: Mapping from operations to their corresponding FDB transactions, i.e., $\sigma(o)$ is the transaction containing operation o

Locking Mechanism Definitions. FDBKeeper uses three types of locks to control operation execution order:

- $RL(p, c)$: Shared read lock held by client c on path p
  - This is a shared lock; multiple clients can hold read locks on the same path simultaneously
  - If operation o satisfies $\tau(o) = \text{READ}$ and $\gamma(o) = c$, it requires read locks on $\pi(o)$ and all its ancestor paths
- $WL(p, c)$: Exclusive write lock held by client c on path p
  - This is an exclusive lock; only one client can hold a write lock on a path at a time
  - If operation o satisfies $\tau(o) = \text{WRITE}$ and $\gamma(o) = c$, it requires a write lock on $\pi(o)$ and read locks on all its ancestor paths
- $CL(c)$: Commit lock for client c
  - Implemented as a FIFO queue to ensure operations from the same client are committed in the order they were issued
  - All operations from client c must acquire and release this lock in sequence
- D_c: Directed Acyclic Graph (DAG) of operation dependencies for client c
  - An edge $o_i \to o_j$ indicates that operation o_j must wait for operation o_i to release at least one lock
  - This graph is used to avoid deadlocks and ensure operations execute in the correct order
- $locks(o)$: Set of all locks required by operation o, including read locks, write locks, and the commit lock

State Function.

- $f : \mathcal{P} \to Values$: Represents the system state, tracking the current value of each path
  - Initially, for all $p \in \mathcal{P}$, $f(p) = 0$ (indicating the path does not exist or has an initial value)
  - The state function is updated by WRITE operations; when operation o executes, if $\tau(o) = \text{WRITE}$, then $f(\pi(o)) = val(o)$

Client Queue.

- $Q_c = \langle q_1, q_2, \ldots \rangle$: Queue of operations from client c, ordered by their issue time

Definition of Linearizability. A history H is linearizable if and only if:

(1) $<_H$ respects real-time order: $\forall o_i, o_j \in O : o_i <_{rt} o_j \Rightarrow o_i <_H o_j$
(2) Executing operations in order $<_H$ produces results consistent with a valid sequential execution

10.3 Construction of Global Linear History H

We now construct FDBKeeper's global linear history H, which reflects the execution order of all operations in the system. This construction shows how FDBKeeper combines FDB's strict serializability capability with its own locking mechanism to achieve linearizability guarantees equivalent to ZooKeeper.

10.3.1 Cross-Client Linearizability via FDB Strict Serializability. In FDBKeeper, each operation is encapsulated as an FDB transaction, forming the transaction sequence $S = \langle t_1, t_2, \ldots, t_m \rangle$. FDB uses a single sequencer to assign a globally unique commit number to each transaction, ensuring strict serializability. This is similar to how ZooKeeper's Zab protocol achieves linearizability for cross-client updates.

Formally, if the end time of operation o_i is earlier than the start time of operation o_j (i.e., $T_e(\sigma(o_i)) < T_s(\sigma(o_j))$), then o_i's transaction $\sigma(o_i)$ must precede o_j's transaction $\sigma(o_j)$ in the FDB transaction sequence S.

10.3.2 FIFO Client Order via Client-Side Locking. FDB itself does not guarantee that operations from the same client are executed in the order they were issued. For example, if a client sequentially sends `Create(/a)`, `Create(/a/b)`, and `Create(/a/b/c)`, FDB does not guarantee they will execute in this order, potentially causing later operations to fail due to dependency violations.

FDBKeeper addresses this issue through its three-type client-side locking mechanism:

(1) **Read Locks (*RL*)**: Shared locks that allow multiple operations to read the same path simultaneously
(2) **Write Locks (*WL*)**: Exclusive locks that ensure serialized write access to paths
(3) **Commit Locks (*CL*)**: FIFO queues that ensure operations from the same client are committed in the order they were issued

For example, consider the operation sequence `Create(/a)`, `Create(/a/b)` and `Create(/a/b/c)`:

- `Create(/a)` requires a read lock on the root path "/" and a write lock on "/a"
- `Create(/a/b)` requires read locks on "/" and "/a", and a write lock on "/a/b"
- The dependency graph D_c ensures that `Create(/a/b)` executes only after `Create(/a)` completes

We define H as the sequence of operations derived from the FDB transaction sequence S, where if $S[i] = \sigma(o)$ then $H[i] = o$. The commit lock mechanism ensures that operations from the same client appear in H in the order they were issued, i.e., according to $<_c$.

10.4 Verification of Linearizability Properties

We now prove that the constructed global linear history H satisfies all properties of linearizability.

10.4.1 Existence and Well-Definedness of Global Order H.

Proposition 10.2. *H is a well-defined total order on O.*

Proof. The FDB transaction sequence S is a total order because FDB's sequencer assigns a unique commit number to each transaction. Since each transaction t_i maps to a unique operation o_i (in our model, each transaction contains one operation), H inherits the total ordering property of S.

For client c, the commit lock $CL(c)$ ensures that the commitment order of its operations matches their issue order $<_c$, while the dependency graph D_c ensures that hierarchical dependencies (e.g., `Create(/a)` $\rightarrow$ `Create(/a/b)`) are respected.

Therefore, $H = \langle o_1, o_2, \ldots, o_n \rangle$ forms a total order on all committed operations. □

10.4.2 Real-Time Order Guarantee (Cross-Client Linearizability).

Proposition 10.3. $\forall o_i, o_j \in O : o_i <_{rt} o_j \Rightarrow o_i <_H o_j$

Proof. Assume $o_i <_{rt} o_j$, meaning operation o_i completes before operation o_j begins in real time ($T_e(o_i) < T_s(o_j)$).

Since each operation is encapsulated as an FDB transaction, we have:

- $T_e(\sigma(o_i)) = T_e(o_i)$ (transaction end time equals operation end time)
- $T_s(\sigma(o_j)) = T_s(o_j)$ (transaction start time equals operation start time)

Therefore, $T_e(\sigma(o_i)) < T_s(\sigma(o_j))$, meaning transaction $\sigma(o_i)$ completes before transaction $\sigma(o_j)$ begins.

According to FDB's strict serializability property, when transaction $\sigma(o_i)$ completes before transaction $\sigma(o_j)$ begins, $\sigma(o_i)$ must precede $\sigma(o_j)$ in the transaction sequence S. Suppose $\sigma(o_i) = t_p$ and $\sigma(o_j) = t_q$, then $p < q$ in S.

By the construction of H, we have $H[p] = o_i$, $H[q] = o_j$, and $p < q$, which means $o_i <_H o_j$.

This proves that H respects the real-time order, corresponding to ZooKeeper's cross-client linearizability guarantee. □

10.4.3 Client FIFO Order Guarantee.

Proposition 10.4. $\forall o_i, o_j \in O : o_i <_c o_j \Rightarrow o_i <_H o_j$

Proof. Consider two operations o_i and o_j from the same client c, where $o_i <_c o_j$ (meaning o_i was issued before o_j).

Let the operation queue for client c be $Q_c = \langle q_1, q_2, \ldots \rangle$, where $q_i = o_i$ and $q_j = o_j$, with $i < j$ (positions in the queue).

FDBKeeper's commit lock $CL(c)$ is implemented as a FIFO queue, ensuring that operations acquire the lock in the order they appear in Q_c. Specifically, operation q_k must commit and release $CL(c)$ before operation q_{k+1} can acquire it, leading to $T_e(q_k) < T_s(q_{k+1})$.

Additionally, the dependency graph D_c enforces operation dependencies. For instance, if q_i creates path "/a" and q_j creates "/a/b", there exists an edge $q_i \rightarrow q_j$ in D_c, ensuring that q_j must wait for q_i to complete.

By induction from i to $j-1$, we can prove that $T_e(o_i) < T_s(o_j)$, meaning $o_i <_{rt} o_j$. According to the real-time order guarantee proven in the previous section, this implies $o_i <_H o_j$.

This demonstrates that FDBKeeper implements the same client FIFO order guarantee as ZooKeeper, addressing FDB's lack of native FIFO support. □

10.4.4 Consistency and Order Validity.

Proposition 10.5. *H ensures consistent ordering of conflicting operations and correct results, including handling of concurrent non-conflicting operations.*

Proof. We examine different operation scenarios:

Conflicting Operations. Consider two operations o_i and o_j that access the same path ($\pi(o_i) = \pi(o_j) = p$) where at least one is a write operation ($\tau(o_i) =$ WRITE or $\tau(o_j) =$ WRITE).

Case 1: Conflicting operations from the same client

If $\gamma(o_i) = \gamma(o_j)$ (same client), then by client FIFO order, either $o_i <_c o_j$ or $o_j <_c o_i$. According to the client FIFO guarantee, this ensures $o_i <_H o_j$ or $o_j <_H o_i$.

Case 2: Write-write conflicts from different clients

If $\tau(o_i) = \tau(o_j) =$ WRITE and they come from different clients, they require exclusive write locks $WL(p, \gamma(o_i))$ and $WL(p, \gamma(o_j))$ on the same path p.

These write locks are mutually exclusive, meaning only after one operation releases the write lock can the other operation acquire it. Specifically:

- If o_i acquires the lock first, it must complete and release the lock before o_j can acquire it, resulting in $T_e(o_i) < T_s(o_j)$

- Conversely, if o_j acquires the lock first, we have $T_e(o_j) < T_s(o_i)$

By the real-time order guarantee, this ensures either $o_i <_H o_j$ or $o_j <_H o_i$, meaning conflicting write operations have a definite order in the global history H.

For example, if client 1 executes `Set('/a', 'value1')` and client 2 executes `Set('/a', 'value2')`, FDBKeeper's write lock mechanism ensures these operations cannot execute concurrently but are executed in some definite order.

Case 3: Read-write conflicts from different clients

If $\tau(o_i)$ = WRITE and $\tau(o_j)$ = READ (or vice versa) from different clients, the write operation requires an exclusive write lock $WL(p, \gamma(o_i))$, while the read operation requires a shared read lock $RL(p, \gamma(o_j))$.

Since write locks and read locks are mutually exclusive (write locks exclude all other locks), this ensures that read and write operations cannot execute concurrently. Therefore, either $T_e(o_i) < T_s(o_j)$ or $T_e(o_j) < T_s(o_i)$, guaranteeing $o_i <_H o_j$ or $o_j <_H o_i$.

Read-Write Consistency. For a read operation $r \in O$ with $\tau(r)$ = READ and $\pi(r) = p$, we need to prove that its return value reflects the most recent value of p according to the order H.

We track the system state using function f: following the order H, for each write operation w, we update $f(\pi(w)) := val(w)$.

Define:

- $W_p = \{w \in O \mid \tau(w) = \text{WRITE}, \pi(w) = p, w <_H r\}$: all write operations on path p that precede r in H
- $last(r) = \max_{<_H} W_p$ (if W_p is empty, $last(r) = \emptyset$): the last write operation on p that precedes r in H

The write lock WL prevents concurrent writes to p, ensuring that r can see the effects of all preceding write operations at the time $T_s(r)$.

According to FDB's transaction semantics and FDBKeeper's locking mechanism:

- If $W_p \neq \emptyset$, then $result(r) = val(last(r))$, meaning r returns the value from the most recent write
- If $W_p = \emptyset$, then $result(r) = 0$ (initial value, indicating the path doesn't exist or is uninitialized)

For example, if history H contains the sequence `Set('/a', 'v1')`, `Get('/a')`, `Set('/a', 'v2')`, then `Get('/a')` will return 'v1', reflecting the result of the last write operation preceding it in H.

Concurrent Non-Conflicting Operations. For non-conflicting operations o_i and o_j (i.e., $\pi(o_i) \neq \pi(o_j)$ or both are read operations), FDBKeeper allows concurrent execution.

- If $\pi(o_i) \neq \pi(o_j)$, they operate on different paths and don't compete for the same locks
- If $\tau(o_i) = \tau(o_j)$ = READ and $\pi(o_i) = \pi(o_j)$, shared read locks RL allow them to acquire locks concurrently

The specific order of these operations is determined by FDB's transaction system, reflected in S and the derived H. Since they don't conflict, their relative order in H does not affect the state function f or the consistency of operation results. □

10.5 Conclusion

Through the above proof, we have demonstrated that the global linear history H satisfies all requirements of linearizability:

(1) **Real-time Order**: $\forall o_i, o_j \in O : o_i <_{rt} o_j \Rightarrow o_i <_H o_j$
(2) **Validity**: H corresponds to a sequential execution where read operations return the values of the most recent write operations

FDBKeeper successfully implements the same two basic guarantees as ZooKeeper:

- **Cross-Client Linearizability**: Through FDB's single sequencer and strict serializability mechanism
- **Client FIFO Order**: Through its three-type client-side locking mechanism (read locks RL, write locks WL, commit locks CL) and dependency graph D_c

These two mechanisms together ensure the linearizability of operations, even though FDB itself does not directly support client FIFO order. Therefore, Theorem 10.1 holds: FDBKeeper guarantees the linearizability of operations by combining the strict serializability of FDB with client-side locking schemes. □