Design Document: Fault-Tolerant Distributed Lock Manager (3-Node SMR)

Name: Rovin Singh(24290010),Naveen Tiwari(24210068)

1. Goals & Requirements

- **Mutual Exclusion (Safety):** Ensure at most one client holds the logical lock at any time. Operations requiring the lock are permitted only for the valid, current lock holder.
- **Liveness:** Guarantee lock acquisition progress for well-behaved clients if the lock is available or becomes available, assuming cluster majority and client availability.
- **Fault Tolerance:** Tolerate a single server crash-stop failure while maintaining service availability and correctness.
- **Idempotency:** Ensure client operations with side effects are processed at most once, despite client retries.
- **Client Crash Handling:** Reclaim locks from unresponsive clients via a lease mechanism.

2. System Model & Assumptions

- **Nodes:** N clients; 3 uniquely identified server nodes (ID 1, 2, 3).
- **Network:** Asynchronous (delay, loss, duplication); partitions possible.
- **Server Failures:** Crash-recovery model (stop, restart, recover state); no Byzantine behavior.
- **Client Failures:** Crash or unresponsiveness.
- **Shared Filesystem:** Reliable, consistent shared storage accessible by all servers for resource data and persisted lock/operation state.

  Clocks: Loosely synchronized (sufficient for leases).

3. Architecture: 3-Node Replicated State Machine

- **Replicated State:** {currentHolderID: int32, currentToken: string, leaseExpiryTime: Time}.
- **Roles:** Single Primary (processes client writes), two Replicas (receive state updates, detect failure, participate in elections).
- **Quorum:** Majority of 2 nodes required for Primary operations and successful elections.
- **Initial State:** Server 1 starts as Primary.

4. Core Mechanisms

4.1. Lock Management & Leases

Time-bound lock ownership granted by Primary with unique tokens. Leases automatically expire if not renewed by the client. Strict token/lease validation by Primary for all lock-protected operations.

4.2. State Replication

Primary asynchronously sends lock state updates {holder, token, expiry} to Replicas after local state changes. Replicas apply updates locally and persist.

4.3. Failure Detection & Election

Replicas heartbeat Primary. On persistent failure, a Replica initiates election if quorum ($\geq$2 live nodes) exists. Lowest live server ID wins.

4.4. Fencing

Newly elected Primary enters a fencing period (LeaseDuration + Buffer), rejecting writes. Exits by clearing persisted lock state (ForceClearLockState) before serving.

4.5. Primary Quorum Check & Step-Down

Primary verifies quorum before writes. Steps down to Replica if quorum is lost.

4.6. Persistence

Atomic writes to lock_state.json (shared FS) for lock state. WAL + processed_requests.log (shared FS) for file append durability and idempotency.

5. Protocol Specification (Brief)

5.1. Client → Primary:

- **Acquire(ClientID, ReqID)** → Response(Status, Token)
- **Release(ClientID, ReqID, Token)** → Response(Status)
- **RenewLease(ClientID, ReqID, Token)** → Response(Status)
- **FileAppend(ClientID, ReqID, Token, File, Data)** → Response(Status)

Behavior: Generate unique ReqID. Target presumed Primary. Retry on timeout/error. Failover to next server on connection error, SECONDARY_MODE, or SERVER_FENCING. Invalidate local token on INVALID_TOKEN.

5.2. Replica → Primary:

Ping(ServerID) → Response(Status) (Heartbeat)

5.3. Primary → Replica:

UpdateSecondaryState(State{Holder, Token, Expiry}) → Response(Status) (Asynchronous Replication)

5.4. Server → Server (Internal/Election):

Implicit Ping calls used during checkQuorum (Primary) and election quorum verification (Replica).

5.5. Server Responses:

- **OK:** Success.
- **ERROR:** Generic failure.
- **INVALID_TOKEN/PERMISSION_DENIED:** Client lacks valid lock ownership/lease.
- **SECONDARY_MODE:** Request sent to a Replica. Client must failover.
- **SERVER_FENCING:** Request sent to a newly elected, fencing Primary. Client must wait/failover.

6. Design Trade-offs

Simplicity vs. Robustness (SMR vs. Raft/Paxos): The simple SMR with lowest-ID election is easier to implement but lacks the formal log consistency guarantees of full consensus protocols, making it potentially more vulnerable in complex partition/recovery scenarios not explicitly covered by fencing.

Performance vs. Consistency (Async Replication): Asynchronous replication yields higher throughput and lower latency but allows replicas' state to temporarily lag the primary. Fencing mitigates the safety risks during failover associated with this lag. Strong consistency would severely impact performance.

Availability vs. Consistency (Quorum): Requiring quorum ensures consistency (prevents split-brain) but makes the system unavailable if a majority of nodes fail or are partitioned.

7. Persistence Summary

Lock State: Atomically persisted (lock_state.json) on shared FS upon every state change by the Primary. Loaded by all nodes on startup.

File Appends: Durability via WAL; Idempotency via processed_requests.log (both on shared FS).