Zookeeper

Architecture -

1. Each server extends a single-node database server and adds fault tolerance using ZooKeeper and Cassandra.
2. Clients connect to any server over TCP; servers do not communicate directly with each other.
3. ZooKeeper is treated as a shared, replicated log under the path /zk_requests, while Cassandra stores durable application state and metadata per replica.

Managing state -

1. Each replica uses a Cassandra keyspace named after the server (server0, server1, server2).
2. Application data is stored in table grade. Replication metadata is stored in table zk_meta.
3. On startup, the server creates these tables if needed, reads last_applied, and may reset it to 0 if the grade table is empty so it can safely reprocess operations from ZooKeeper after test resets.

Order and replication -

1. Every client CQL request is appended to ZooKeeper as a persistent sequential znode under /zk_requests .
2. The numeric suffix of the znode name is treated as the global sequence number. All servers periodically read and sort the children of /zk_requests and apply operations in ascending sequence order.
3. The combination of sequential znodes and strict ordered replay ensures that all replicas see the same sequence of updates.

Sequential Apply Logic -

1. A background executor and ZooKeeper watches both call applyPendingRequests.
2. The method then:
    a. Fetches /zk_requests children with a watch, sorts them by sequence, and skips any with seq <= last_applied.
    b. Computes the next expected sequence and processes requests in strict order, up to a safety iteration limit to avoid starvation.
    c. For each request, it Reads the CQL from ZooKeeper, qualifies table names with the server's keyspace if needed. Executes the CQL on Cassandra; on success, increments and persists last_applied and releases any waiting client. If there is an execution error, it stops and leaves last_applied unchanged so the same sequence will be retried later.

Log Clean up -

1. To keep ZooKeeper from growing indefinitely, gcOldRequestsIfNeeded deletes older znodes with sequence numbers <= last_applied once the log exceeds a configured size

If the ZooKeeper session expires, the watcher detects KeeperState.Expired and closes the server so it can be restarted cleanly. Because ordering is driven entirely by ZooKeeper's sequential znodes, any server that has replayed up to the same last_applied will have the same logical application state in its grade table.

GIGA-Paxos

Architecture -

1. MyDBReplicableAppGP implements Replicable and is instantiated by Gigapaxos .
2. Each replica connects to a single Cassandra instance on 127.0.0.1:9042 using a keyspace passed in args[0].
3. Gigapaxos ensures that all replicas see the same ordered sequence of RequestPackets.

Request Execution -

- execute(Request, boolean):
    - Casts to RequestPacket and reads requestValue (string).
    - Parses messages of the form reqID::CQL or just CQL.
    - Treats all non-empty CQL strings as queries directly against Cassandra. No separate read/write paths; both read and write queries are simply executed via session.execute(query).

Checkpointing -

1. Reads the entire grade table from Cassandra.
2. Serializes the logical state into a compact string
3. This string is returned to Gigapaxos as the checkpoint image

Restore -

1. If checkpoint is empty, it just returns true.
2. Otherwise, it serializes state string back into (id, events) pairs
3. And then, re-inserts rows into grade with appropriate events lists.

Failure Handling -

1. When a replica crashes and restarts:
    a. Gigapaxos re-creates MyDBReplicableAppGP with the same keyspace.
    b. Then, calls restore with the last checkpoint string to rebuild the grade table.
    c. Replays the Paxos log entries after that checkpoint by re-invoking execute in the original order.