9. Consistency and Consensus

- Consistency Guarantees
  - Most replicated DBs provide at least eventual consistency
    - Aka "convergence"
    - But reading your own writes is not guaranteed
  - Eventual consistency is inconvenient for developers and often need more guarantees
    - But they come at some price of performance or less fault-tolerance
  - Consistency overlaps but is different from isolation levels (transactions)
- Linearizability
  - Each business operation should clearly define what type of isolation and consistency is needed!
  - Linearizability - illusion of single replica
    - Aka "atomic consistency" aka "strong consistency" aka "immediate consistency" aka "external consistency"
    - It is a "recency guarantee"
  - What Makes a System Linearizable?
    - Linearizability vs Serializability
  - Relying on Linearizability
    - Locking and leader election
    - Constraints and uniqueness guarantees
    - Cross-channel timing dependencies
  - Implementing Linearizable Systems
    - Replication types
      - Single-leader replication
        - Read from leader or sync-follower
          - Potentially linearisable
        - Read from async
          - Can violate durability and linearisability
      - Consensus algorithms
        - linearizable
          - Thanks to consensus algorithms
      - Multi-leader replication
        - not linearizable
      - Leaderless replication
        - probably not linearizable
    - Linearizability and quorums
      - Can fix problem with cost of performance
        - Read repair
        - ? Writer should read latest state of quorum
      - But last-write-wins still breaks linearizability
  - The Cost of Linearizability
    - network interruption causes app to become unavailable
    - The CAP theorem
      - The Unhelpful CAP Theorem
        - Consistency, Availability, Partition tolerance: pick 2 out of 3
        - Network partitions will happen anyway! So you don't have choice
      - either Consistent or Available when Partitioned
    - Linearizability and network delays
      - Even RAM in multicore CPU is not linearizable!
        - To increase performance
- Ordering Guarantees
  - Ordering and Causality
    - Ordering is importance because it helps preserve causality
    - Ordering = causally consistent
    - The causal order is not a total order
      - Linearizability
        - =total order
      - Causality
        - =partial order
          - Two events can be incomparable if they happen concurrently
    - Linearizability is stronger than Causal Consistency
      - causal consistency is the strongest possible consistency model that
        - does not slow down due to network delays
        - and remains available in the face of network failures
    - Capturing causal dependencies
      - Version vectors can be generalized to track dependencies across the entire database
  - Sequence Number Ordering
    - Sequence number can provide total order!
      - Can be timestamp or counter
    - Noncausal sequence number generators
      - But how generate them in distributed systems?
        - All approaches are not fully consistent with causality
    - Lamport timestamps
      - consistent with causality
      - Node id + Counter
        - Increases counter for each request
        - If client sees larger counter it updates it's counter
    - Timestamp ordering is not sufficient
      - Can create the same user name twice
      - Need to know when order is finalized
  - Total Order Broadcast
    - Aka "atomic broadcast"
    - Scope of ordering guarantee
    - Two properties are required
      - Reliable delivery
        - No messages are lost
      - Totally ordered delivery
        - Messages delivered in the same order to each node
    - Using Total Order Broadcast
      - It is exactly what is needed for db replications
        - state machine replication
      - Can be used for serialisable transactions
      - Can be used for logs
      - Can be used for lock services
    - Implementing Linearizable storage using Total Order Broadcast
      - Total Order Broadcast does not answers WHEN message is delivered to different nodes
        - One recipient may lag behind another
      - But Linearizability also guarantees recency
        - Read is guaranteed to see the latest write
      - Can implement linearizability using total order broadcast
        - Write
        - Read my write - ensure it is the first
    - Implementing total order broadcast using linearizable storage
- Distributed Transactions and Consensus
  - intro
    - Consensus
      - get several nodes to agree on something
    - Situations
      - Leader election
      - Atomic commit
    - The Impossibility of Consensus
      - FLP result
        - No also to reach consensus if nodes may crash
        - But if we use clocks and timeouts to detect crashes then it is solvable
  - Atomic Commit and Two-Phase Commit (2PC)
    - Atomicity is especially important for
      - Multi-object change
      - Secondary indexes
    - From single-node to distributed atomic commit
    - Introduction to two-phase commit (2PC)
      - XA transactions
        - supported by the Java Transaction API
      - WS-AtomicTransaction
        - for SOAP web services
      - Don't confuse 2PC and 2PL
      - Uses coordinator or transaction manager
    - A system of promises
      - 2 points
        - "yes" answer of the node
          - It promises to commit if it is asked
        - Decision made by coordinator
          - Which he will later try (with forever retries) to deliver to each node
    - Coordinator failure
      - After node answered "yes" it should wait for coordinator's decision
        - Transaction is "in doubt" or "uncertain"
    - Three-phase commit (3PC)
      - 2PC is blocking
        - Can stuck, waiting coordinator to recover
      - 3PC is non-blocking
        - But requires network with bounded delays and nodes with bounded response times
          - i.e. it requires perfect failure detector
          - Which is impractical
            - So we keep using 2PC
  - Distributed Transactions in Practice
    - Bad performance
    - Two types
      - Database-internal distributed transactions
      - Heterogeneous distributed transactions
    - Exactly-once message processing
    - XA transactions
      - X/Open XA (short for eXtended Architecture)
        - Standard implementation of 2PC for heterogeneous technologies
    - Holding locks while in doubt
      - Other requests may be locked if they try to read row which is in doubt
    - Recovering from coordinator failure
      - "orphaned" in-doubt transactions do occur
      - The only way - administrator manually decide: commit or abort
        - But it is manual and stressful work when everything is bad
      - Many XA implementations have automated Emergency escape
        - heuristic decisions
          - probably breaking atomicity
    - Limitations of distributed transactions
      - If any part fails the transactions also fail
        - So - distributed transactions are amplifying failures
  - Fault-Tolerant Consensus
    - Properties
      - Uniform agreement
        - No two nodes decide differently
      - Integrity
        - No node decides twice
      - Validity
        - If a node decides value v, then v was proposed by some node
      - Termination
        - Every node that does not crash eventually decides some value
          - e.g. 2PC does not meet this requirement!
    - Consensus algorithms and total order broadcast
      - Best algos
        - Viewstamped Replication (VSR)
        - Paxos
        - Raft
        - Zab
      - Most of algos decide on sequence
        - Which makes them Total Order Broadcast algos
          - Nodes propose a message they want to be sent next
          - Nodes decide on the next message to be delivered
    - Single-leader replication and consensus
      - In order to solve consensus (reelect the leader) we need to have a leader?
    - Epoch numbering and quorums
      - Each leader is assigned epoch
        - If two leaders exist then the one with the highest epoch wins
      - Two rounds of voting
        - One to choose the leader
        - Second to vote for leader's proposal
        - Voting nodes must overlap with at least one node!
    - Limitations of consensus
      - Performance is not good
      - Good algo is still an open question
  - Membership and Coordination Services
    - projects
      - ZooKeeper
      - etcd
    - Features of ZooKeeper
      - Linearizable atomic operations
      - Total ordering of operations
      - Failure detection
      - Change notifications
    - Allocating work to nodes
      - Useful for selecting new leader
      - But also also useful for
        - job schedulers and similar stateful systems
        - partitioned resource when need to rebalance
      - Apache Curator
        - A higher level tool on top of ZooKeeper
      - Zookeeper is for slow changing data
        - If need fast changing
          - Apache BookKeeper
    - Service discovery
      - ZooKeeper, etcd, and Consul
      - Can also help other services discover who the leader is
        - Via read-only caching replicas
    - Membership services
      - Can use consensus to select who is currently alive (members)
        - And for example can select leader by lowest ID among members
- Summary
  - You need strong consistency (Linearizability)
    - But it is expensive
  - You can usually use cheaper and weaker consistency - Causal Consistency
    - But sometimes it is not enough
      - e.g. you can end up creating the same login concurrently by different users
  - You can solve such problems by Consensus
    - but it requires Linearizability
      - Which is expensive
      - cycle? 😅
  - Good news are
    - We do not need Consensus for EVERY operation
      - We can use single leader with Linearizability
        - And Consensus for selecting new leader
      - Or we can use Causal Consistency
        - And Consensus for some problem cases
    - We have specific software that implements Consensus
      - e.g. ZooKeeper
  - Also some systems (leaderless / multileader) don't need linearizability
    - Can work with branching and merging