Design and Engineering of Computer Systems

Lecture 37: Replication and Consistency

Mythili Vutukuru
IIT Bombay
Replication and Consistency

- One way to build fault tolerant systems: **replicated state machines**
  - Requests from clients (inputs) are processed by all replicas
  - All replicas start with same state, handle same input, so will stay in the same state always (assuming deterministic processing)
  - Used to build active-active replicated systems
  - Used to build reliable distributed data stores for active-passive systems

- Challenge in building replicated state machines: **consistency**
  - What if a replica was down and didn’t receive some input?
  - What if a replica received some inputs in a jumbled order?
  - How to ensure all replicas are always consistent, i.e., in the same state?

- This lecture: mechanisms for replication that guarantee consistency
Consistency models

• Many definitions / models of consistency, some strong and some weak

• Atomic consistency is example of strong consistency model
  • All inputs / operations (e.g., add/delete/view items in shopping cart) executed at all replicas in exactly the same order
  • If an operation $Y$ (e.g., view shopping cart) starts after operation $X$ (e.g., add item to cart) finishes according to some global clock, then $Y$ should always see the result of $X$

• Eventual consistency is example of a weak consistency model
  • If an operation $Y$ (e.g., view shopping cart) starts after operation $X$ (e.g., add item to cart) finishes, then $Y$ should see the result of operation $X$ eventually

• Spectrum of consistency models from strong to weak
  • Example: causal consistency model says that same order of operations/inputs maintained only between operations that impact each other (e.g., operations on same shopping cart) and not across all operations
How to achieve strong consistency

• Requires the use of **consensus protocols**: messages exchanged among replicas to let them agree on certain decisions

• **Raft**: popular, widely used consensus protocol that lets multiple replicas agree on an consistent ordered log of entries
  • Example: replicas of shopping cart servers agree on a log containing various operations (add/delete/view cart) to be performed on shopping carts

• Building a replicated state machine with Raft
  • All replicas run Raft, agree on a consistent replicated log containing the same operations in the same order at all replicas
  • After an operation replicated in log using Raft, all replicas execute the operation, stay consistent with each other

• Other consensus protocols can be used to build replicated state machines, e.g., **Paxos** lets all replicas reach agreement on a single value in each round
Strong consistency: Raft (1)

- Basic idea of Raft: replicas exchange messages to maintain a consistent replicated log (same entry at same index in every log)
  - Replicas elect one leader, rest are followers
  - Leader receives inputs from clients, propagates to all replicas in the form of log entries
  - Once leader has replicated entry at majority of nodes, entry considered committed, applied to state machine, confirmation returned to client
  - What if majority of replicas cannot be contacted? No response to client
  - Example of a quorum protocol: contact a quorum before returning response
  - Raft instance with $2f+1$ replicas can tolerate up to $f$ failures
  - Leader failure: followers detect via heartbeats, elect new leader, start new term (old leader can come back up and join as follower later)
Strong consistency: Raft (2)

- Replica failures can cause logs to diverge
  - Some follower may have briefly crashed and missed a few log entries
  - Old leader of previous term can have some extra uncommitted entries that it did not manage to replicate before it crashed

- How does leader reconcile such logs?
  - When leader propagates entry $k$, it also mentions its entry “$k-1$”. Follower updates entry “$k$” only if its entry “$k-1$” matches with that of leader
  - If a follower’s previous entry does not match, leader will rollback to the point where logs match and help follower catch up with all previous committed entries
  - Leader tries to sync all follower’s logs to its own log

- Leader’s log is the authoritative source, so it is crucial to elect good leader
  - All replicas vote for node with most up to date log (with all committed entries)
  - Leader elected successfully only if it gets majority of votes ($f+1$ out of $2f+1$)
  - Any two majorities always intersect, so at least one node with up-to-date log of previous term will be available to be elected as leader in next term
Weak consistency: Dynamo (1)

• Raft and other strong consistency protocols: if majority cannot be contacted, no response returned to client, system is unavailable sometimes

• What if we want high availability? What if we return a response back even if client request is not replicated at a majority of nodes?
  • One client request (add item to cart) replicated only at a minority of nodes (due to failures)
  • Another request (view cart) executed at another minority of nodes
  • It is possible to have two minority sets with no intersection, so viewing cart may not reflect latest item added to cart
  • Inconsistent values can be returned by the system, but service always available

• Some systems accept weak consistency in return for high availability, e.g., Dynamo NoSQL key-value store (Amazon) has high availability, only eventual consistency
  • Sloppy quorum protocol, response returned to client even if replication not successful at all desired replicas due to replica failures
  • Systems eventually tries to catch up the missing replicas, but no guarantees on timelines
Weak consistency: Dynamo (2)

• Systems with weak consistency can have conflicting values of application state
  • Shopping cart of user has items A, B
  • User adds item C, replicated only at a minority of nodes (due to failures)
  • User adds item D, replicated at a different minority of nodes
  • When user views cart, can get back “A, B, C” or “A, B, D” or both
  • Note that this can never happen with Raft: at least one node will have seen both updates, as any two majorities will intersect

• Application can decide how to handle inconsistent values
  • Merge shopping carts to have superset of all items “A, B, C, D”
  • Trickier to merge two different versions of bank accounts
Which replication / consistency model to use?

• How to replicate, how much consistency depends on application needs
  • Online banking server may prefer a strong consistency model
  • Shopping cart server may be okay with a weaker consistency model

• In general, providing stronger consistency models requires more work in the application, and higher performance overheads

• Tradeoff between consistency and availability
  • Providing strong consistency requires contacting majority of replicas
  • What if some replicas cannot be reached, say due to network partition/failure?
  • Systems providing strong consistency will become unavailable at such times
  • Systems with weak consistency will be available but may return inconsistent results

• **CAP theorem**: you can get only two but not all out of (strong) Consistency, (high) Availability, (network) Partition tolerance
Summary

• In this lecture:
  • Strong and weak consistency models
  • Different ways of replication to achieve strong/weak consistency
  • Tradeoff between consistency and availability, CAP theorem

• Refer to the original papers on Raft and Dynamo:
  • "In Search of an Understandable Consensus Algorithm", Ongaro and Ousterhout.
  • "Dynamo: Amazon's Highly Available Key-value Store", DeCandia et al.