Midterm 2 Review

Midterm Topics

- Leader Election
- Consensus
  - Formulation
  - Synchronous consensus
- Paxos
- FLP Theorem
- Bitcoin
- Raft
- ~~DHT~~

Midterm Format

- Timed exam, 7–9 p.m. Monday
- Exam released @ 7 p.m.
- Upload your answers to Gradescope by 9 p.m.
  - Typed file strongly preferred
  - Scanned handwritten solutions accepted in a pinch
- Open book but no collaboration (honor code)
- Zoom meeting (office hour) for clarifications
  - Google Doc for shared clarifications

Leader Election

- Goal: elect a single leader
  - Upon detecting current leader failure
  - Tolerate multiple elections
  - Pick leader based on attribute
- Properties
  - Safety: each process considers the unique live process with highest ID new leader
  - Liveness: a leader is eventually elected

Leader Election Algorithms

- Ring
  - Send election message around the ring
  - Keep track of highest ID seen
  - If highest seen ID = mine, consider self elected
  - One more trip around to tell everyone else
- Bully
  - Send messages to all processes with higher IDs
  - If none respond (timeout) consider self elected
  - Otherwise, those processes start election themselves

Consensus

- Each process has some input value x_i value and output value y_i
- Goal: all processes agree on some output value
  - Once output is set, process is in decided state, output cannot change
- Properties:
  - Termination: Each (correct) process reaches decided state
  - Agreement: If two correct processes p_i and p_j reach decided state, their outputs must match, $y_i = y_j$
  - Integrity: If all processes have the same input, they must use it as the output: $x_i = x_j$ for all i, j => $x_i = y_i$

Synchronous Consensus

Tolerates up to f failures

- Round 1: B-multicast value to all other processes
- Round 2…f+1: B-multicast all [newly] received values to all other processes
- Decision: pick smallest of all received values

Intuition: after a failure-free round, all processes have the same set of values

FLP Theorem

Impossible to have totally correct consensus in asynchronous systems

- Important since consensus is equivalent to many other things
  - Leader election
  - Totally ordered multicast
  - …
- In practice: design so that guarantees are met if communication delay within a bound
- Otherwise, give up on liveness (eg. Paxos, Raft) or safety

FLP Model

- Configuration: state of all processes + message buffer
  - Initial configuration: processes' inputs + all initial messages
- Event
  - Deliver a message from message buffer to one process
  - Update state & generate new messages
- Schedule: sequence of events
- Model: all messages *eventually* delivered but can be *arbitrarily* delayed, reordered
- Configurations:
  - 0-Valent: result in decision of 0
  - 1-Valent: result in decision of 1
  - Bivalent: can result in either decision

FLP Proof Steps

Lemma 1: Schedules that involve non-overlapping processes commute

Lemma 2: There exists a bivalent initial configuration

Lemma 3: From a bivalent configuration, another bivalent configuration is always reachable

Conclusion: there is an infinite path of bivalent configurations, i.e., never reaches decision => liveness not satisfied

[despite all messages being eventually delivered]

Paxos: Consensus Protocol

Phase 1:

- Proposer sends a *prepare* request with a proposal #
- Acceptor replies with:
  - Promise not to reply to proposals with lower #
  - Promise not to accept proposals with lower #
  - # and value of highest # proposal accepted
- No reply if would violate previous promise

Phase 2:

- If majority of acceptors reply, proposer sends accept request
  - Value = highest among replies from acceptors
- Acceptor accepts proposal if would not violate promise
- Signals acceptance to learners

Learners:

- Decision reached once majority of acceptors accept the same value

Log consensus

- State machines: some process state that gets updated in response to events
- Replicated state machines
  - Maintain availability / durability if some processes fail
- Log consensus: agree on exact sequence of events/updates that get applied
- Log consensus + deterministic state machines => consistent replicated state machines

Raft consensus

Leader election

- Started when leader heartbeat timeout expires (randomized)
- Starts a new *term*

A node votes for the leader if

- Has not voted for anyone else in same term
- Candidate's log is up to date

Majority votes: candidate elected

Split elections possible, term incremented

Log replication

- Leader acts as sequencer for events
- Replicates events to followers
- Event replicated by a majority of followers can be *committed*

Log consistency

- Logs of followers get *overwritten* to match leader's log
- Majority restriction + up-to-date-check: committed entries never get overwritten

Bitcoin / Blockchain

Designed for log consensus among large-scale, dynamic groups

Transaction ordering:

- Group transactions into blocks
- Chain blocks using hash functions

Gossip-based broadcast

- Transactions / blocks gossiped to random neighbors
- Viral spread across the network

Block creation:

- Decentralized “leader election” by solving verifiable puzzle
  - Difficulty tuned to limit block creation rate
- Chain split / forks possible, but long-lived forks unlikely
- k-deep transactions are considered committed
  - E.g., k=6